ZeroTier: The Internet As A Global Switch

Terragrunt: The Orchestrator Tool For Terraform

₹120
ISSN-2456-4885

Volume: 10 | Issue: 03 | Pages: 108 | January 2022

OpenSource ForYou

THE COMPLETE MAGAZINE ON OPEN SOURCE

EFY GROUP Publication

Smart Ways To Connect Devices

Using Node-RED For IoT And Edge Computing

Balena: Helping To Manage IoT Fleets

An Introduction To IoT Protocols

Do You Have A Strategy To Manage Risks In Open Source Web Applications?

Want To Write Quality Code? Try Logging In Python

www.opensourceforu.com

Wanna Be Your Own Boss?

DO OPEN SOURCE.

Demand for Open Source is sky rocketing. Be it for managing IT infrastructure or development of software--Open Source solutions are what customers are seeking.

All you need to do is develop expertise in an Open Source stack, and then build a team around it!

And, Open Source For You can be your friend and a guide through this journey.

TO READ OUR PRINT EDITION Visit: https://subscribe.efyindia.com

TO READ OUR EZINE EDITION Visit: https://ezine.lfymag.com

WORLD'S LEADING PUBLICATION ON OPEN SOURCE

Looking for marketing solutions to engage with cutting edge techies?
Contact us at growmybiz@efy.in OR call us at +91-9811155335.

Does Your Antivirus Solution Provide You With Complete Protection?

Don't RELY on 20 years old technology to FIGHT current date viruses

TOP 10 things that your Antivirus solution should provide to tackle today's threat...

Ransomware File Protection	ATP- Advance Threat Protection	EDR- Endpoint Detection And Response	
Application Blocking	Deep Learning Malware Analysis	Exploit Prevention	URL Blocking
Disk and Boot Record Protection	Peripheral Control	Respond Investigate Remove (Root Cause Analysis)	

If it doesn't, contact us for a solution which can

Contact: Santosh on 9971696319 or Email at santosh.gupta@itsipl.com

ITS
A Total Solutions Company

I. T. Solutions India Private Limited
D-88/5, Okhla Industrial Area, Okhla Phase I, New Delhi -110020
Ph: 011-47695000 • Email: sales@ itsipl.com • www.itsipl.com

- Mumbai
- Jaipur
- Chandigarh

SOPHOS 2020 BEST NEXTGEN PARTNER | Delhi/ NCR
SOPHOS 2019 BEST NEXTGEN PARTNER | Delhi/ NCR
SOPHOS 2018 BEST SI PARTNER | Delhi

CONTENTS

JANUARY 2022 | ISSN-2456-4885

FOCUS

39 Choose the Most Optimal IoT Protocol for Your Project

44 Using Node-RED for IoT and Edge Computing

48 An Introduction to IoT Protocols

DEVELOPERS

62 Managing Secrets with SecretsFoundry

67 Implementing Simple Linear Regression Using Python

72 User Interfaces in R

79 How Python Can Help to Process Audio Waves

86 Try These Unusual Unicode Tricks

COLUMNS

95 CodeSport

ADMIN

97 Terragrunt: The Orchestrator Tool for Terraform

99 ZeroTier: The Internet as a Global Switch

104 Remote Working and BYOD: Can Virtual Desktop Infrastructure Secure Your Enterprise?

12 **Careers:** Building a Career in the IoT-Led World

15 Rewarding Contributors, Building Role Models: The FSF Awards

The Design Odyssey

18 Exploring Behavioural Patterns for **Asynchronous Communication**

56 **Want to Write Quality Code? Try Logging in Python**

83 Stop Arguing and Start Cashing In on Open Source Specifications

REGULAR FEATURES

07 FossBytes

EFYGROUP
YOURS SINCE 1969

Wanna Support Open Source For You?

Subscribe to the Magazine— so that we can keep promoting Open Source.

AMAZING OFFER

Pay for 6 Issues Get 12!

(Buy One Get One Free!)

WORLD'S LEADING PUBLICATION ON OPEN SOURCE

TEAR OUT — **ORDER FORM** — TEAR OUT

Please Tick (✓) Your Choice

☐ **Pay for 12 Issues Get 24 Issues**
and save 50%
(Buy 12 Issues Get 12 Issues Free!)
₹1440

☐ **Pay for 24 Issues Get 48 Issues**
and save 50%
(Buy 24 Issues Get 24 Issue Free!)
₹2880

☐ **Pay for 36 Issues Get 72 Issues**
and save 50%
(Buy 36 Issues Get 36 Issue Free!)
₹4320

To subscribe online, visit
https://tinyurl.com/y5kuv4la

OR SCAN THIS CODE

Name_____ Organisation_____ Mailing Address_____

_____ City_____

Pin Code_____ State_____ Phone No._____ Email_____

Subscription No. (for existing subscribers only)_____. **I would like to subscribe to the above (✓)marked Open Source For You magazine starting with the next issue.** Please find enclosed a sum of Rs_____ by DD/MO/crossed cheque bearing the No._____ dt._____ in favour of **EFY Enterprises Pvt Ltd,** payable at Delhi. *(Please add Rs 50 on non-metro cheque)*

Please mark one (nearest) relating to your subscription: ☐ Indian Company ☐ MNC ☐ R&D organisation ☐ Engineering institute ☐ College/School ☐ Any other (specify):_____

Send this filled-in form or its photocopy to : **EFY Enterprises Pvt Ltd,** D-87/1 Okhla Industrial Area, Phase 1, New Delhi 110 020 | Ph: **011-40596600** | e-mail: **support@efy.in**

Terms:- # These rates are applicable for new subscribers as well as renewal by existing subscribers. # Can access ezine till your subscription is active # The rates are valid for subscribers within India only. # Please allow 4-6 weeks for processing of your subscription. # The subscription copies will be dispatched through ordinary post only # Subscription Agents will not get agency commission against this scheme # Disputes, if any, are subject to exclusive jurisdiction of competent courts and forums in Delhi/New Delhi only. * Replacement will be made if intimation of damaged / non-receipt of copies is received within 30 days of its publication ** After three months, if you are not satisfied with the magazine, your balance amount will be returned (Not applicable for gift offer)

EDITOR
RAHUL CHOPRA

EDITORIAL, SUBSCRIPTIONS & ADVERTISING
Delhi (HQ)
D-87/1, Okhla Industrial Area, Phase I, New Delhi 110020
Phone: +91-9811155335
E-mail: info@efy.in

MISSING ISSUES
Phone: +91-9811155335
E-mail: support@efy.in

BACK ISSUES
Phone: +91-9811155335
E-mail: support@efy.in

NEWSSTAND DISTRIBUTION
Phone: +91-9811155335
E-mail: efycirc@efy.in

ADVERTISEMENTS
NEW DELHI (HEAD OFFICE)
Phone: +91-9811155335
E-mail: efyenq@efy.in

MUMBAI
E-mail: rmowest@efy.in

BENGALURU
E-mail: rmosouth@efy.in

CHINA
Worldwide Focus Media
E-mail: china@efy.in

GERMANY
pms Plantenberg Media Service GmbH
E-mail: germany@efy.in

JAPAN
Tandem Inc.
E-mail: japan@efy.in

TAIWAN
J.K. Media
E-mail: taiwan@efy.in

UNITED KINGDOM
ASA Media
E-mail: uk@efy.in

UNITED STATES
E & Tech Media
E-mail: usa@efy.in

Printed, published and owned by Ramesh Chopra. Printed at Tara Art Printers Pvt Ltd, A-46,47, Sec-5, Noida, on 28th of the previous month, and published from D-87/1, Okhla Industrial Area, Phase I, New Delhi 110020. Copyright © 2021. All articles in this issue, except for interviews, verbatim quotes, or unless otherwise explicitly mentioned, will be released under Creative Commons Attribution-NonCommercial 3.0 Unported License a month after the date of publication. Refer to http://creativecommons.org/licenses/by-nc/3.0/ for a copy of the licence. Although every effort is made to ensure accuracy, no responsibility whatsoever is taken for any loss due to publishing errors. Articles that cannot be used are returned to the authors if accompanied by a self-addressed and sufficiently stamped envelope. But no responsibility is taken for any loss or delay in returning the material. Disputes, if any, will be settled in a New Delhi court only.

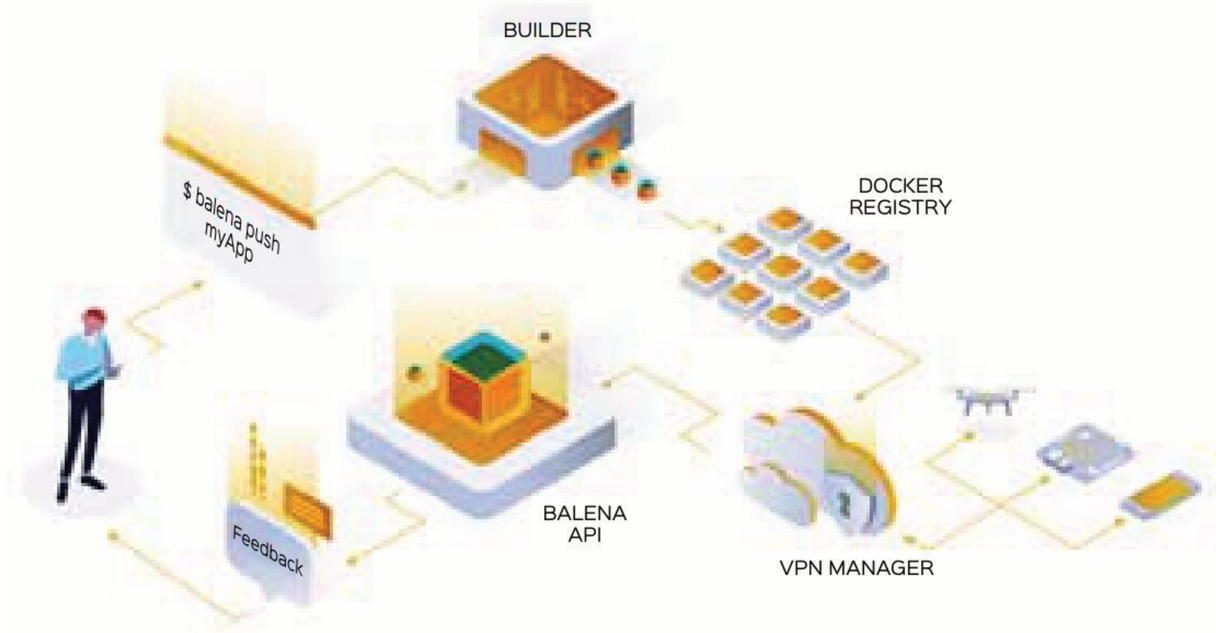
SUBSCRIPTION RATES			
Year	**Newstand Price (₹)**	**You Pay (₹)**	**Overseas**
Five	7200	4320	—
Three	4320	3030	—
One	1440	1150	US$ 120

Kindly add ₹ 50/- for outside Delhi cheques.
Please send payments only in favour of EFY Enterprises Pvt Ltd.
Non-receipt of copies may be reported to **support@efy.in**—do mention your subscription number.

CONTENTS

24

Balena: Helping to Manage IoT Fleets



32

Keep the Fish Healthy with this IoT Based Aquarium

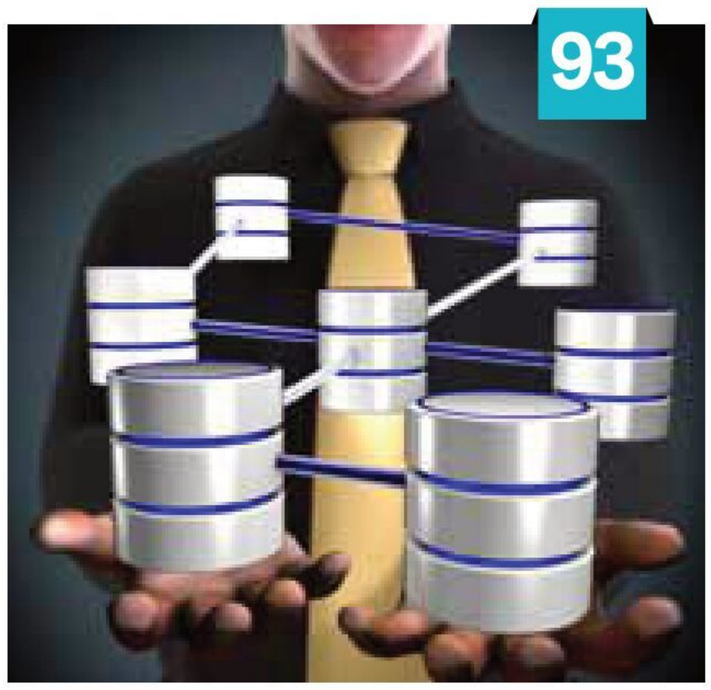
53

Installing and Using Node-RED on Ubuntu

91

Do You Have a Strategy to Manage Open Source Risks in Web Applications?

93



When is NoSQL the Best Choice for Database Management?

PostgreSQL is now available on the Instaclustr platform

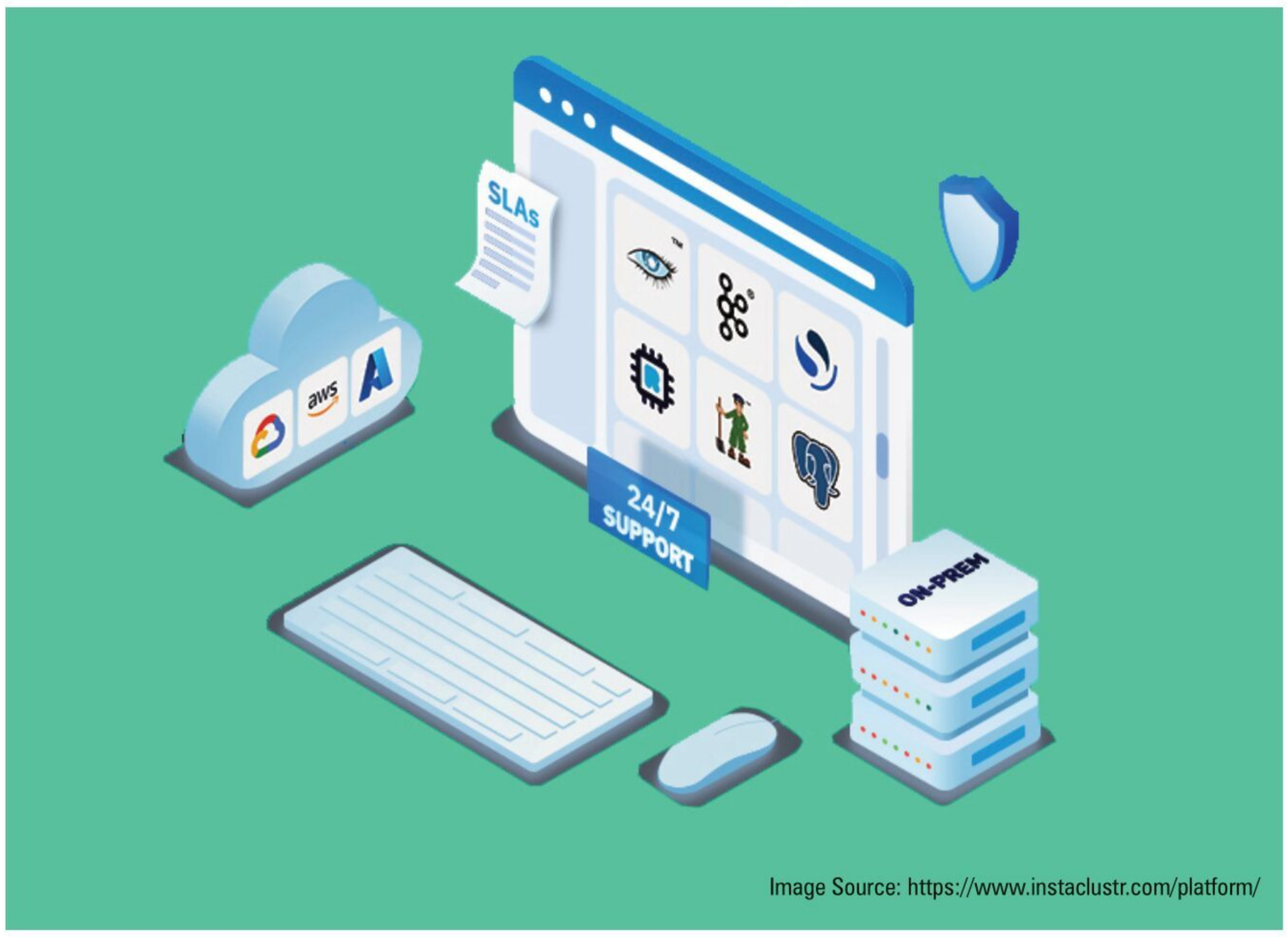


Image Source: https://www.instaclustr.com/platform/

Instaclustr has announced the general availability of PostgreSQL on the Instaclustr platform. Backed by full service PostgreSQL management, which includes secure and end to end database monitoring, optimising, and scaling, customers gain the advantages of open source Postgres without the risk of proprietary lock-in or expensive licensing. The general availability of Postgres on the Instaclustr platform comes after the completed public preview of the service for existing Instaclustr customers in 2021. Postgres joins other fully open source solutions – including Apache Cassandra, Redis, Apache Kafka, and Open Search – as part of a complete ecosystem for customers' data infrastructure needs.

While Postgres' ease of entry also contributes to its rapid adoption, the database becomes considerably more complex to manage at scale – requiring continuous optimisation to maintain performance, security, cost-efficiency, and availability. In Postgres deployments involving mission-critical enterprise environments, proactive and continual monitoring and tuning are required to ensure proper operation.

Instaclustr for PostgreSQL is cheaper and eliminates the risk common to organisations that operate the database internally. The managed solution can provide production-ready Postgres clusters in minutes and, as part of Instaclustr's platform, Instaclustr for PostgreSQL includes 24×7 support and SLAs. It also delivers PostgreSQL in its 100 per cent open source version, continuing Instaclustr's open source commitment to ensuring that customers always have complete control over their code and data, and total freedom from vendor or technical lock-in.

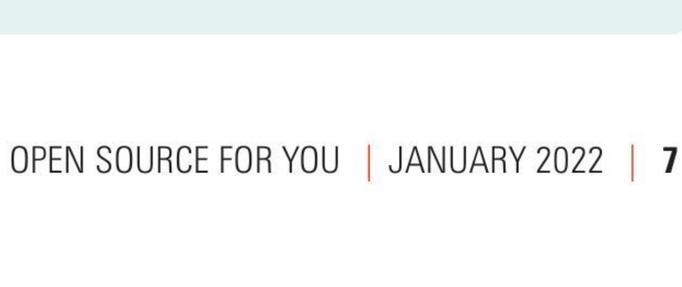
This general availability of Instaclustr for PostgreSQL comes just months after Instaclustr's acquisition of credativ, the global consulting company delivering comprehensive services and technical support for Postgres and other open source data infrastructure technologies.

AWS announces the release of the AWS cloud development kit

AWS has released the AWS cloud development kit (AWS CDK), which is an open source software development platform for defining cloud application resources in programming languages that are familiar to users.

It includes constructs, which are high-level components that preconfigure cloud resources with known defaults, that assist in quickly designing cloud applications. AWS CDK uses AWS CloudFormation to provision resources in a secure and repeatable manner. It also allows creation and distribution of customised structures that incorporate company-specific requirements, leading to faster completion of projects.

CDK structures for defining Terraform HCL state files in TypeScript and Python are provided by cdktf for users who prefer Terraform. The cdk8s project enables Kubernetes users to leverage CDK structures in TypeScript, Python, and Java to define Kubernetes settings. cdk8s can also be used with the AWS CDK's Amazon Elastic Kubernetes Service (Amazon EKS) build library to define Kubernetes infrastructure

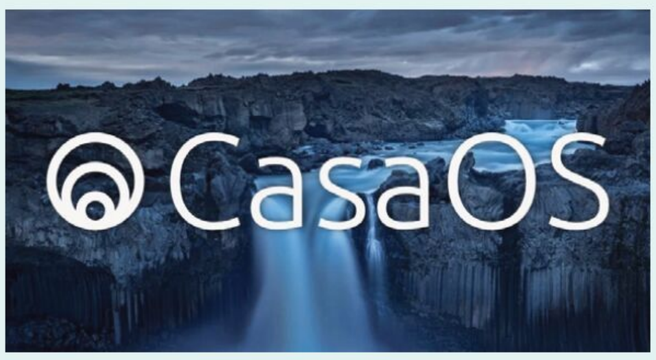


running everywhere. The alpha versions of cdk8s and cdktf are both available. Construct Hub, a location to discover and exchange construct libraries published by the open source community, AWS, and partners, is a good place to start if you want to find all of these CDKs in one spot.

IceWhale Technology announces CasaOS, an open source home cloud OS

IceWhale Technology has announced CasaOS, an open source home cloud OS based on the Docker ecosystem and designed for home scenarios.

Founded in 2020, the company is harnessing the potential of blockchain technology. CasaOS is a new home cloud OS designed to create a decentralised family cloud category. CasaOS will let users store photos and videos. It will also allow them to make encrypted communications and manage smart home devices.



CasaOS is compatible with mainstream hardware platforms, x86 PCs, NUCs, and Raspberry Pi. Its users can host various home entertainment, streaming, network enhancement, and VPN applications free of charge, as well as centralise their personal data and sync much faster in a non-inductive LAN.

"We focus on home scenarios to provide consumers with home cloud product innovation that integrates software and hardware. At a time when the world is exploring more economical and greener infrastructure, we believe that compared to public clouds, home clouds can provide higher system energy efficiency, data security, and more personalised digital services for local home smart experiences," said Lauren Pan, founder and CEO of IceWhale Technology.

Soda launches the Cloud Metrics Store for data health

Soda, the open source data reliability tools provider, has released the Cloud Metrics Store. Available to all users of Soda's open source tools, the store captures historical information about the health of data to support its intelligent testing across every workload.

According to the company, without a clear strategy to monitor data for quality issues, many organisations fail to catch the problems that can leave their systems exposed and result in serious downstream issues. Inspired by modern software engineering principles, Soda is giving data teams the tools to create a culture and community of good data practices through a combination of the Soda Cloud Data Observability platform and its OSS data reliability tools, built by and for data engineers.

With this latest OSS release, Cloud Metrics Store gives data and analytics engineers the ability to test and validate the health of data based on previous values. These historical metrics allow data tests to use a baseline understanding of what good data looks like, with any bad data efficiently quarantined for inspection before it impacts data products or downstream consumers. Alerts are sent via popular on-call tools or Slack, so that data teams are the first to know when data issues arise, and can swiftly resolve the problem.

Soda's data reliability tools work across the data product life cycle. All checks can be written 'as-code' in an easy-to-learn configuration language. Configuration files are version controlled, and used to determine which tests to run each time new data arrives into a data platform.

"It's advantageous for data teams to unify around a common language that allows them to specify what good data looks like across the data value chain from ingestion to consumption, irrespective of roles, skills, or subject matter expertise. Most data teams are organised by domain, and when creating data products, they often depend on each other to provide timely, accurate, and complete data," explained Maarten Masschelein, CEO, Soda.

OSCE supported training course for criminal justice practitioners

A five-day OSCE (Organization for Security and Co-operation in Europe) supported training course on Internet investigations and open source intelligence gathering for criminal justice practitioners, organised by the five Central Asia OSCE participating states, concluded on December 10, 2021.

The training course was based on materials developed by the European Cybercrime Training and Education Group (ECTEG). It emphasised counter-terrorism through case studies and practical exercises, while also addressing themes like open source intelligence gathering methodologies and tradecraft, social network investigations, online intelligence search and collection, and human rights compliance in cybercrime investigations, among others.

The training was given as part of the OSCE's 'Capacity-building on countering cybercrime in Central Asia' initiative, which was carried out by the OSCE Transnational Threats Department in close collaboration with OSCE field operations in the region. The goal is to develop long term national cybercrime training capacities for police personnel and prosecutors.

Two 'train the trainers' courses and topical training courses on managing digital evidence by first responders, as well as investigating crimes enabled by the usage of Dark Web and virtual currencies, were delivered this year.

Kickstarter to support development of new open source protocol

Kickstarter has announced that it is going support the development of an open source protocol that will essentially create a decentralised version of Kickstarter's core functionality. This will live on a public blockchain, and be available for collaborators, independent contributors, and even Kickstarter competitors, from all over the world to build upon, connect to, or use.

"We've chosen Celo, an open source and carbon negative blockchain platform, as the best technology and community on which to build the protocol. We're inspired by the Celo ecosystem's approach to building the technology they want to see in the world," said a company spokesperson.

An autonomous organisation to kickstart protocol development will be set up. Kickstarter PBC will fund this new independent company, select an initial board of directors, and commit to being one of the protocol's first clients. This means Kickstarter.com will be constructed on top of the protocol. Users will continue to have the same Kickstarter experience that they are used to. Although they will not be able to see the protocol, they will benefit from the improvements that will be made.

There is still a lot of work to be done on this project. Kickstarter also intends to establish a governance lab, which will be separate from both the existing and new companies, and provide independent governance for the protocol. The purpose of creating yet another separate corporation is to prevent Kickstarter from exerting undue influence over what is supposed to be a mostly decentralised operation.

Linux Foundation to host the Cloud Hypervisor project

The Linux Foundation has announced it will host the Cloud Hypervisor project, which delivers a virtual machine monitor for modern cloud workloads. Written in Rust with a strong focus on security, its features include CPU, memory and device hot plug; support for running Windows and Linux guests; device offload with vhost-user; and a minimal and compact footprint.

In an official announcement, the Foundation said the project is supported by Alibaba, ARM, ByteDance, Intel, Microsoft, and many others. The focus of the project will initially be on security and modern operation for the cloud.

"Cloud Hypervisor has grown to the point of moving to the neutral governance of The Linux Foundation," said Arjan van de Ven, Intel fellow and founding technical sponsor for the project. "We created the project to provide a more secure and updated VMM to optimise for modern cloud workloads. With fewer device

Critical vulnerability found in open source tool Apache Log4j

A recently discovered cybersecurity vulnerability in Apache Log4j, an open source software program used by a considerable number of enterprises, could potentially allow hackers to install malware on impacted systems. Security specialists from around the world have raced to fix this vulnerability which is being deemed as one of the worst computer flaws in many years — a serious hole in open source code that is used widely in cloud services and corporate software by industry and government.

The Apache Software Foundation, which supervises Log4j development, has released a patch for this newly discovered flaw and hopes to have this issue fixed as soon as possible. If downloading the patch isn't possible anyhow, the organisation has also provided the required guidelines on how users might avoid this flaw. Users have been asked to deploy the patch to the affected systems right away by the US Cybersecurity and Infrastructure Security Agency.

Log4j comes with several prominent open source programs managed by the Apache Software Foundation, and the vulnerability poses a very significant cybersecurity risk. These tools help to power a large number of apps around the world, many of which may now be more vulnerable as a result of this flaw. Apple Inc.'s iCloud cloud storage service and Microsoft Corp.'s popular Minecraft video game are thought to be among the affected products.

CNCF announces 18 new members at Open Source Summit China 2021

The Cloud Native Computing Foundation (CNCF), which develops ecosystems for cloud native software, has announced the addition of 18 new Silver members, non-profit organisations, and end user supporters at the Open Source Summit China 2021. New members will join ranks with leading technology innovators and end users to contribute to the creation, diffusion, and adoption of cloud native technologies.

Cloud native computing helps organisations to build and run applications with an open source software stack in public, private, and hybrid clouds. The Cloud Native Computing Foundation (CNCF) hosts components of the global technology infrastructure, including Kubernetes, Prometheus, and Envoy.

Adoption of cloud native technology has been increasing rapidly in China over the last several years. According to the Cloud Native Survey China 2020, 68 per cent of organisations in the country use containers in production, which is a 39 per cent increase over 2019 and a 240 per cent increase from two years ago. This trend is likely to accelerate, as Chinese organisations have been progressively deploying containers in test environments.

"CNCF has almost 70 member organisations headquartered in China and now hosts 25 projects that were born in China including Dragonfly, KubeEdge, Longhorn, and TiKV," said Priyanka Sharma, general manager of the Cloud Native Computing Foundation. "We are excited to reconvene virtually with our Chinese community, and are thrilled to be announcing 21 new members to join our diverse and innovative #TeamCloudNative."

With the addition of these new members, there are now over 160 organisations in the CNCF end user community.

models and a modern, more secure language, Cloud Hypervisor offers security and performance optimised for today's cloud needs."

"Modern cloud workloads require better security, and the Cloud Hypervisor project is intentionally designed to focus on this critical area," said Mike Dolan, senior vice president and general manager of projects at the Linux Foundation.

K.Y. Srinivasan, advisory board member from Microsoft, added: "Cloud Hypervisor has matured to the point that moving it to the Linux Foundation is the right move at the right time. As LF continues to standardise key components of the software stack for managing/orchestrating modern workloads, we feel that the Cloud Hypervisor will be an important part of the overall stack. Being part of LF will help us accelerate development and adoption of this key technology."

The European Commission updates guidelines for open source software

The European Commission has unveiled new open source software guidelines that will allow it to release software under open source licences. The decision was made in response to a study by the Commission that revealed that investing in open source software yields four times higher profits on average.

The new rules will increase transparency and help the Commission, as well as citizens, companies, and public services across Europe, benefit from open source software development. This can also enhance security as external and independent specialists check software for bugs and security flaws.

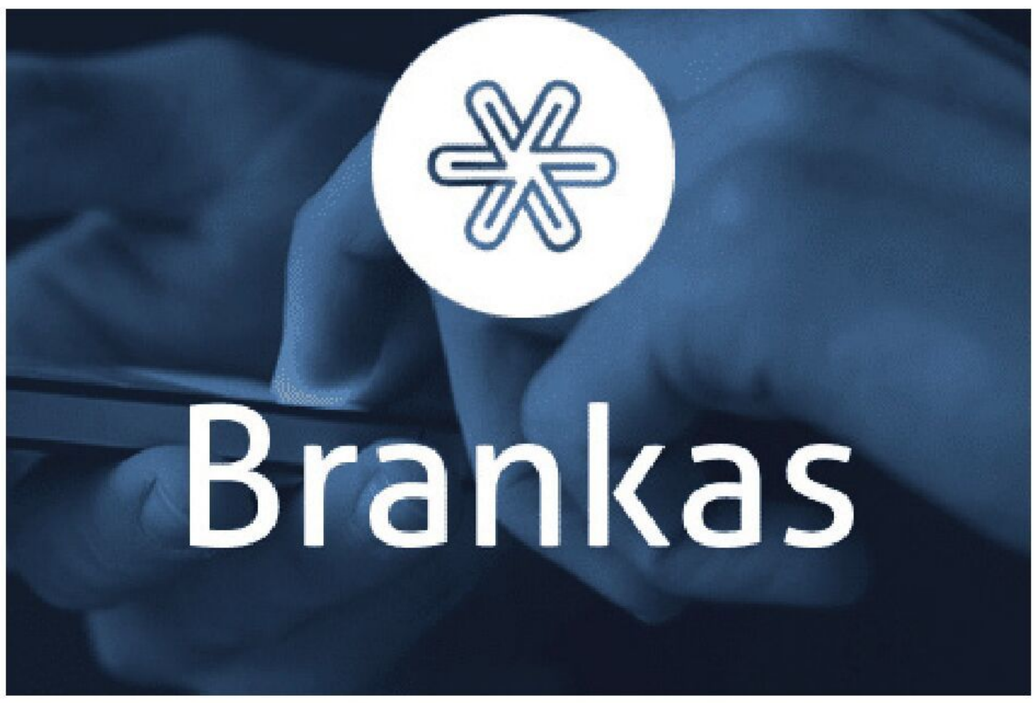
The EC has announced the eSignature, a set of free standards, tools and services that help public administrations and businesses accelerate the creation and verification of electronic signatures that are legally valid in all EU member states. It has also announced LEOS (legislation editing open software), which is used to create legal texts across the Commission. LEOS is being developed in close partnership with Germany, Spain, and Greece, and was originally written for the Commission.

APIX Open Core to be developed further by Brankas

The APIX Open Core banking system is being developed by Brankas for traditional and digital banks, non-bank financial institutions (NBFIs), fintech companies, and cooperatives in emerging markets. With funding support from the FSTI POC grant, Brankas aims to further develop APIX Open Core to enable rapid financial product development. It aims to do so by removing licensing costs, providing easy integration with third-party services, and ensuring compliance with regulatory requirements.

In Singapore, where the technology will be developed and deployed first, local fintech companies will benefit from a faster entry into new markets as adoption of the APIX Open Core expands across the ASEAN region, while the local software engineering community can directly contribute to a 'homegrown' open source project that helps to narrow a persisting gap in the region's financial services infrastructure.

APIX Open Core is an open source core banking system based on the APIX platform. APIX was set up by AFIN (ASEAN Financial Innovation Network) in 2018 as the first cross-border, open-architecture API marketplace for collaboration between fintech and financial institutions. With APIX, participants can integrate and test solutions in a sandbox environment via a cloud based architecture.



"APIX Open Core is designed to disrupt the banking status quo. The aim is to solve current barriers to entry in the banking sector. For instance, commercial CBS from established players tends to be very costly, restricting banks to a closed environment that allows minimal integration and customisability," said Mike Dickinson, chief product Officer of Brankas.

APIX Open Core currently contains eight modules targeted for retail banking. It is also designed to allow integration with external fintech partners, the bank's mobile or Web apps, and external card or payment networks.

Apple revamps its website to let users access open source data

Apple has released a revamped version of its website, which lets users access open source data. Apple's open source initiatives, as well as those of third parties, are highlighted on the new website. The website, which is open to the public, is a location for developers to learn about and contribute to the open source projects that Apple also participates in. It also allows developers to download the open source code for its operating system from GitHub.

'Apple Projects' and 'Community Projects' are the two sections of the new website. The Swift programming language, the WebKit engine for Web browsers, the FoundationDB database, ResearchKit and CareKit for health-related information, and Password Manager Resources are all open source projects that can be found on the new website.

Developers can also find third-party projects in the 'community projects' section, in addition to Apple's own project, which includes open source projects led by groups other than Apple but created with Apple engineers' assistance.

For more news, visit *www.opensourceforu.com*

Open Source MLOps tool adds experiment versioning

Iterative, the MLOps company, has announced the release of data versioning control (DVC) experiment versioning. The latter gives developers an easy method to save, compare, and reproduce ML experiments at scale in ways that neither traditional software version control nor existing experiment tracking tools can.

"Current experiment tracking tools usually provide an API to log experiment information, a database to store it, and a dashboard to compare and visualise. DVC experiment versioning builds on newer version control principles, which allow to address experiment tracking needs and give developers an integrated way to iterate their experiments," said Dave Berenbaum, technical product manager at Iterative, for MLOps.

Experiment versioning is lightweight, using an existing tech stack and eliminating the need for additional services. Automated reproduction saves time and complexity, while distributed and flexible collaboration enables teams to generate experiments individually and share them as they choose.

With experiment versioning, data science teams can:

- Restore or reproduce any experiment automatically
- Log experiments end-to-end and track changes introduced by each
- Keep experiments connected to their Git repo, with no external services needed

DVC provides users with a Git-like interface for versioning data, models, and pipelines, bringing version control to machine learning and solving the challenges of reproducibility. Experiment versioning extends DVC's capabilities beyond simple experiment tracking.

Building a Career in the IoT-Led World

With the booming success of IoT, how can you build a bright career in this field? What are the job roles available here? What skill-sets will help you make your mark? How far can veterans contribute in this field? The answers are all here.

Covid-19 has left the IoT field in shambles as the world has seen massive changes in the way the industry now works. Those organisations who adopted the technology long before the pandemic have seen a rise in their profits, while those that did not adopt have either closed down or, unfortunately, have had to bear the brunt of losses. This shows that for an organisation's survival and growth IoT has became a strategic necessity.

With IoT invading every industry and organisation, the solutions being delivered by solution providers are also creating new job opportunities. With no dearth of opportunities in this field and the relevance to both tech and business realms, a demand for highly skilled IoT professionals seems to be on the rise. "I think it is crucial to understand that IoT is not one technology but a domain or a concept that brings multiple technologies under its umbrella to make something happen at the end of the day," says Vinay Solanki, GM & head of Digital Solutions, Napino Auto & Electronics.

What IoT job profiles look like

During past couple of decades India has emerged as a hotspot for the software industry manned by budding engineers. Through IoT they are now expected to provide solutions to the customers' core issues. IoT is an interface between the customers and the analytics/insights and not just an interface connecting electronic appliances to the Internet to monitor them.

Many job roles have transitioned from software to the IoT industry where every skill-set is equally important and lucrative. Solanki goes on to add, "I think it is a very good takeaway that, apart from learning programming languages or looking at cloud services, if you are building your career right now, and you come from a background which is not computer science, then it's not bad news for you."

Here are some areas and roles budding engineers can look at to build their careers.

Hardware and embedded systems: While most IoT solutions require software, hardware also plays a vital part. Building reliable hardware has become imperative for the overall IoT industry. With the government's focus on making India a manufacturing hub, a ton of jobs may be expected to flow in.

Solution architect: This is a very critical role, as a system integrator needs to have thorough knowledge of all aspects starting from hardware to connectivity. It also requires knowledge of applications to the cloud platform in which a specific IoT application would be posted in.

This role is indispensable in the case of networking and telecom sectors, since connectivity is a vital aspect of an IoT solution. With wide range of options for connectivity such as Wi-Fi, LoRa, Bluetooth, etc, the authority of choosing the best option lies in this role.

Product manager: This role involves managing all parts of a project till it reaches the customer's hands. In the case of manufacturing, where a company is involved in making smart consumer devices, a competent IoT product manager should understand the hardware and have it programmed as

per requirement. His responsibilities include managing the product after deployment along with the content security policies (CSP) to make sure the product is working the way it should. He also needs to coordinate with R&D department and other teams within the organisation to make sure that the right solution is provided through the existing electronics of their product to make it work seamlessly.

Legal advisor: Conformance to numerous legal and regulatory requirements is most important for any IoT product or solution. Constant change in data storage rules of different countries makes it even more necessary to have a legal team that cross-checks all the related laws and requirements before a product is placed in the market.

IoT enabler: As the name suggests, an IoT enabler ensures delivery of time-bound end-to-end solution as per requirement. Sunil David, regional director of IoT, AT&T, says, "It is a very critical role because many times you find projects stuck in the product cycle as they need somebody who is an IoT champion within the organisation who can work inside the organisation."

Lifecycle manager: He manages the solution once it is implemented and makes sure that the solution keeps working well for the customer.

Security officer: Customers are usually afraid of the threats of their IoT system's data being leaked to malicious parties. These threats pose a major need for the data to be properly secured as it may contain sensitive information about the user. According to David, "With a huge shortage in India for cybersecurity professionals, individuals with a career in hardware or software security are most needed."

Data scientists: With complexity in dealing with huge volumes of ever-flowing data, the need for data scientists cannot be overestimated for an organisation. Their role is primarily to prepare meaningful and useful data from the huge volume of data being generated.

Gaps in the industry and skill-sets to develop

Lakshmi Mittra, VP & head, Clover Academy, says, "The diversity in the field of IoT eliminates the need for an individual to be from a specific background." For roles such as system integrator, solution architect, or IoT provider, candidates need to have the skill-set of dealing with sales and businesses as well, not just the technical aspect, because that is where the pain point of the customer lies. No doubt technical skills are what drive the IoT innovation and development, but in today's era of different domains in an organisation intertwining and collaborating, having a wide range of soft skills is just as important, if not more.

Given how IoT solves the day-to-day problems of its customers and aims to make life easier, having a flair for innovative thinking and creative abilities to look at the challenges and come up with the right solution is also essential. Additionally, having a curious mindset that searches for problems and solutions in everyday life is what sets them apart. The next gap to bridge would be the soft skills and need for creativity. "I have seen that people who know how to design innovative mechanical products at the right design parameters, cost and consideration, especially complicated ones like wearables, are mostly in demand because it is a very important skill to have," adds Solanki.

One of the most important skill-sets an aspiring candidate needs to possess is good communication skills. IoT is a domain that thrives on team effort. There are way too many aspects in this game for a single player to master, and it is imperative to have the right team from a product standpoint, putting the complete situation together so that the product is good. Good interpersonal skills are necessary because it requires collaboration and communication. Communication skills can be termed as another gap that has to be bridged.

"I think someone who can articulate and communicate well is very important. Even is at the junior level, people need to build other skill-sets in the industry and not just be technology experts. I think what can take you farther and help you climb up the ladder of career faster is that if you are also good in soft skills," adds Mittra.

Building your career as a fresher

Our education system lags far behind in imparting students with industry-related skills and knowledge. So, interested candidates need to start training themselves through internships, workshops, and courses as it is futile to depend on the system. Students and freshers from such an environment are often worried about how to make it big in the IoT space. Students of Tier 2 or Tier 3 cities are especially confused about how they could gain an edge over their peers from elite institutes.

David notes, "One of the gaps that I see often in such candidates is the lack of industry touch in comparison to the candidates from metro cities. They really don't know the practical aspects of what they are doing, or how they are going to shape their career." Hence, expanding their area of interest and having knowledge of the business aspect of the product or solution is very important. This aspect is something that young professionals tend to ignore as they strive to use their technical expertise alone.

"Apart from the technical skills, I think it is important to have the willingness to learn. The whole attitude is so important," David adds. To take on higher challenges of the domain for any student or fresher, learning and upskilling needs should be the highest priority. For instance, becoming a better coder, learning the marketing aspects of IoT, or familiarising yourself with cloud services like AWS or Microsoft Azure is now easier with the help of the Internet.

Panelists

Dhaval Chhedha, VP, Engineering, Masai School

Lakshmi Mittra, VP & head, Clover Academy

Sunil David, regional director of IoT, AT&T

Vinay Solanki, GM & head of Digital Solutions, Napino Auto & Electronics

Aspiring candidates must make full use of this opportunity. Dhaval Chhedha, VP, Engineering, Masai School, says, "Practical learning is something that you have to develop from within because it's not going to happen. I don't expect our educational system to change anytime soon, so students have to take up projects on their own."

Solanki advises, "Go beyond the four walls of your house to attend events that speak of innovation. Remember the 3 C's: Co-create, Co-invent, and Collaborate. IoT is such a field where no one company or team can build everything. Instances to work with competitors may arise as well. In early career stages, think of how you can collaborate with someone in your college to create a product together because you won't be good at everything."

For a senior professional

This exciting domain is not just limited to new talent; veterans and experienced professionals are needed as well in taking the industry forward. However, the switch from one industry to another, which is so dynamic and constantly evolving, may seem daunting.

Experience is the father of wisdom and, in that matter, someone with varied but still deep experience in one domain is an asset for any company. For instance, an individual who is very strong in sales, or in building partnerships, or managing a team of solution architects, needs to also have basic knowledge of how the other teams function. "With everything converging so fast, building depth in a couple of areas but breadth in many areas and having a T-shaped career is important. A person cannot just be a sales guy, but needs to also have a little knowledge concerning data security," says Solanki.

While the aspiration of senior professionals to enter a new company by leading a particular team might be valid, David notes that it is important to have emphatic leadership. He goes on to say, "If you want to lead the team, you have to empathize with the hard working folks who are on the team. This reverse mentoring is very important."

The lens through which the young professionals look at the world is one bright side of interaction between both senior and young professionals. It helps broaden horizons and understand what is going inside the minds of younger folks. As per Mittra, the ability to put together and run a team is needed in senior folks as it helps build the right team and makes people work on the solution together.

Chhedha says, senior professionals or leaders should be able to hire people who are smarter than themselves as that would not only make their task easier but also help in finding better solutions. However, he also adds, "The important thing in a leader is to know and understand what is happening around them and going past them. If not, things would start crumbling very fast and the leader would be taken for a ride."

Crossing a certain threshold in one's career does not mean that one should stop learning or assume having all the knowledge required to excel. Learning has to be a lifelong process since it will especially help someone trying to enter a new industry or domain. After 5-10 years of experience, it usually gets difficult to break the ice and get back to learning. Given that IoT can be deployed across pretty much any industry, aspiring professionals need not be confined to IT or telecom because there are enough and more opportunities in other industries also.

In conclusion, Solanki adds, "For those building their profile in this field, it is important that you understand the domain very well, and you also understand it is no longer required to work only in sunrise or so-called attractive industries. With the need of more digital people, I think there is no end to making your mark in this industry." END

This article is based on the panel discussion titled 'Career Guidance - What are the Latest IoT Jobs and How to Prepare For Them?' held during June edition of Tech World Congress 2021.

By: Siddha Dhar

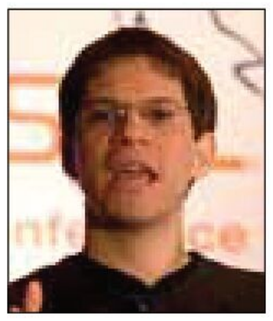
The author is a business journalist at EFY.

The article was originally published in the November 2021 issue of Electronics For You.

Rewarding Contributors, Building Role Models:

The FSF Awards

Bradley M. Kuhn



Miguel de Icaza

Yukihiro Matsumoto

Harald Welte

FREE SOFTWARE FOUNDATION

Alyssa Rosenzweig

Rob Savoye

Wietse Venema

Ted Ts'o

The Free Software Awards were instituted in the late nineties, and have been awarded to individuals and companies that are true contributors to free software.

The Free Software Foundation will be presenting its 2021 awards for the advancement of free software at the LibrePlanet conference in March 2022. These awards are given each year.

The Free Software Foundation (FSF) is a non-profit with a worldwide mission to promote computer user freedom. "We defend the rights of all software users," it says. As a part of its activities, it offers three awards — for the advancement of free software, for a project of social benefit, or for an outstanding new free software contributor.

Earlier awards read like a who's who of contributors to the free software world. In March 2021, the Award for Advancement of Free Software went to CiviCRM, Bradley Kuhn, and Alyssa Rosenzweig.

CiviCRM allows non-profit organisations to manage their mailings and contact databases. It works for the FSF too. Thus, it was seen by the FSF as "allowing charities and non-profit organisations to do their work without being beholden to a particular proprietary software vendor."

Canadian hacker Alyssa Rosenzweig leads the Panfrost project to reverse engineer and provide free firmware for the Mali series of GPUs. An Asahi Linux developer, she aims to port GNU/Linux to the Apple M1 processor. Last July, she demonstrated Debian running bare metal on the Apple M1 with a mainline kernel. She also wrote a GNU/Linux client for the Covid-19 contract tracing used in Canada in September 2020.

Kuhn, meanwhile, has worked long to enforce the GNU General Public License (GPL) and to promote copyleft through the Software Freedom Conservancy (which focuses on "ethical technology").

Other earlier awardees include:

- 2019: Let's Encrypt, a non-profit certificate authority that hopes to make encrypted Web traffic the default state of the entire Internet; the prolific free software programmer-maintainer-writer Jim Meyering; and Clarissa Lima Borges (a young Brazilian doing usability testing on GNOME applications).
- 2018: OpenStreetMap (the collaborative project to create a free editable map of the world); and Deborah Nicholson, who "advocates access to political information, unfettered freedom of speech and assembly, and civil liberties in our increasingly digital world."
- 2017: Public Lab (working to democratise science and address environmental issues); and Karen Sandler, the executive director of the Software Freedom Conservancy.
- 2016: SecureDrop (the anonymous whistle-blower platform, originally written by the famed late Aaron Swartz); and the Brazilian hacker Alexandre Oliva. Oliva is the maintainer of linux-libre, the fully free version of the kernel Linux.
- 2015: Library Freedom Project (LFP) and Werner Koch. The LFP is a partnership among librarians, technologists, attorneys, and privacy advocates working for intellectual freedom in libraries. Koch is the founder of GnuPG, the free software cryptographic suite.
- 2015: Sébastien Jodogne (for free software medical imaging and his project Orthanc); and Reglue, which was giving GNU/Linux computers to underprivileged children and their families in Austin, Texas.
- 2014: Matthew Garrett, from the GNOME Foundation's Outreach Program for Women.
- 2012: Dr Fernando Perez got the award as the creator of IPython, as it provides a rich architecture for interactive computing with an all-in-one debugger, editor, and Python command-line interpreter.
- 2011: Yukihiro 'Matz' Matsumoto received the award as the creator of the Ruby programming language.
- 2010: Given to Rob Savoye, a hacker who helped dozens of projects including GCC, GDB, DejaGnu, Newlib, Libgloss, Cygwin, eCos, Expect, multiple major GNU/Linux distributions, and One Laptop Per Child.
- 2009: The award for the Advancement of Free Software was won by John Gilmore, a founder of Cygnus Solutions, which gave free software a place in the business world early on before it gained popularity. The award for Project of Social Benefit was won by the Internet Archive, whose co-founder and chairman is Brewster Kahle.
- 2008: Wietse Venema and Creative Commons. Venema, a Dutch programmer and physicist, is known for writing the Postfix email system, and the TCP Wrapper. The FSF noted: "Since its launch in 2001, Creative Commons has worked to foster a growing body of creative, educational and scientific works that can be shared and built upon by others. Creative Commons has also worked to raise awareness of the harm inflicted by increasingly restrictive copyright regimes."
- 2007: Harald Welte, the German programmer, founder of Osmocom and involved with netfilter/iptables and Openmoko. He is also a member of the Chaos Computer Club. Osmocom, the open source mobile communication project, implements multiple mobile communication standards — GSM, DECT, TETRA, etc.
- 2006: Ted Ts'o, known for his contributions to the Linux kernel and to file systems (e2fsprogs, ext2, ext3, and ext4).
- 2005: Andrew Tridgell, the Australian programmer, author and contributor to the Samba file server (a free re-implementation of the SMB networking protocol), and co-inventor of the rsync algorithm for transferring and synchronising files.
- 2004: Theo de Raadt, the South African, Canada based founder of OpenBSD and OpenSSH, and a founding member of NetBSD.

Earlier winners of the Free Software Awards include Alan Cox (2003, for advocating software freedom, his outspoken opposition to the US' DMCA and other technology control measures, and work on the Linux kernel).

Also honoured were individuals whose work impacted technology and knowledge flows at the global level, including Lawrence Lessig (2002, for promoting an understanding of the political dimension of free software, including the idea that "code is law"), Guido van Rossum (2001, for inventing and implementing as free software the Python programming language), and Brian Paul (2000, for his "ground-breaking work on the Mesa 3D Graphics Library").

Miguel de Icaza (the 27-year-old Mexican) got the award in 1999 for his work on the GNOME project, and a year earlier (1998) Larry Wall received it for Perl, a robust scripting language for sophisticated text manipulation and system management.

What is surprising is that India's contribution is not visible here. END

By: Frederick Noronha

The author is an independent journalist and publisher, and has written on FOSS for almost two decades.

EFY GROUP
YOURS SINCE 1969

Stay **Connected.**
Stay **Informed.**
Stay **Ahead.**

SUBSCRIBE AND **SAVE**

ORDER FORM

PRINT MAGAZINE	1 YEAR (12 copies each)	3 YEARS (36 copies each)	5 YEARS (60 copies each)
		WITHIN INDIA (IN RUPEES)	
Electronics For You (Rs 100/copy)	840 ☐	2150 ☐	3000 ☐
		SAARC COUNTRIES (IN US$ BY AIR)	
	50 ☐	135 ☐	NA
		OTHER COUNTRIES (IN US$ BY AIR)	
	100 ☐	270 ☐	NA

e-magazine subscriptions within India are available at half the rates mentioned here.
Overseas rates for each e-magazine in US$: 1 year: $12; 3 years: $33; 5 years: $50 only

PRINT MAGAZINE SUBSCRIBERS GET:
- Free e-magazine every month
- Free delivery of print magazine by post
- And much more (check: subscribe@efy.in)
- For delivery by courier, please add Rs 50 for each copy

To subscribe online, visit https://payment.efyindia.com | **OR** | **SCAN THIS CODE**

Name_____ Organisation_____ Mailing Address_____

_____ City_____

Pin Code_____ State_____ Phone No._____ Email_____

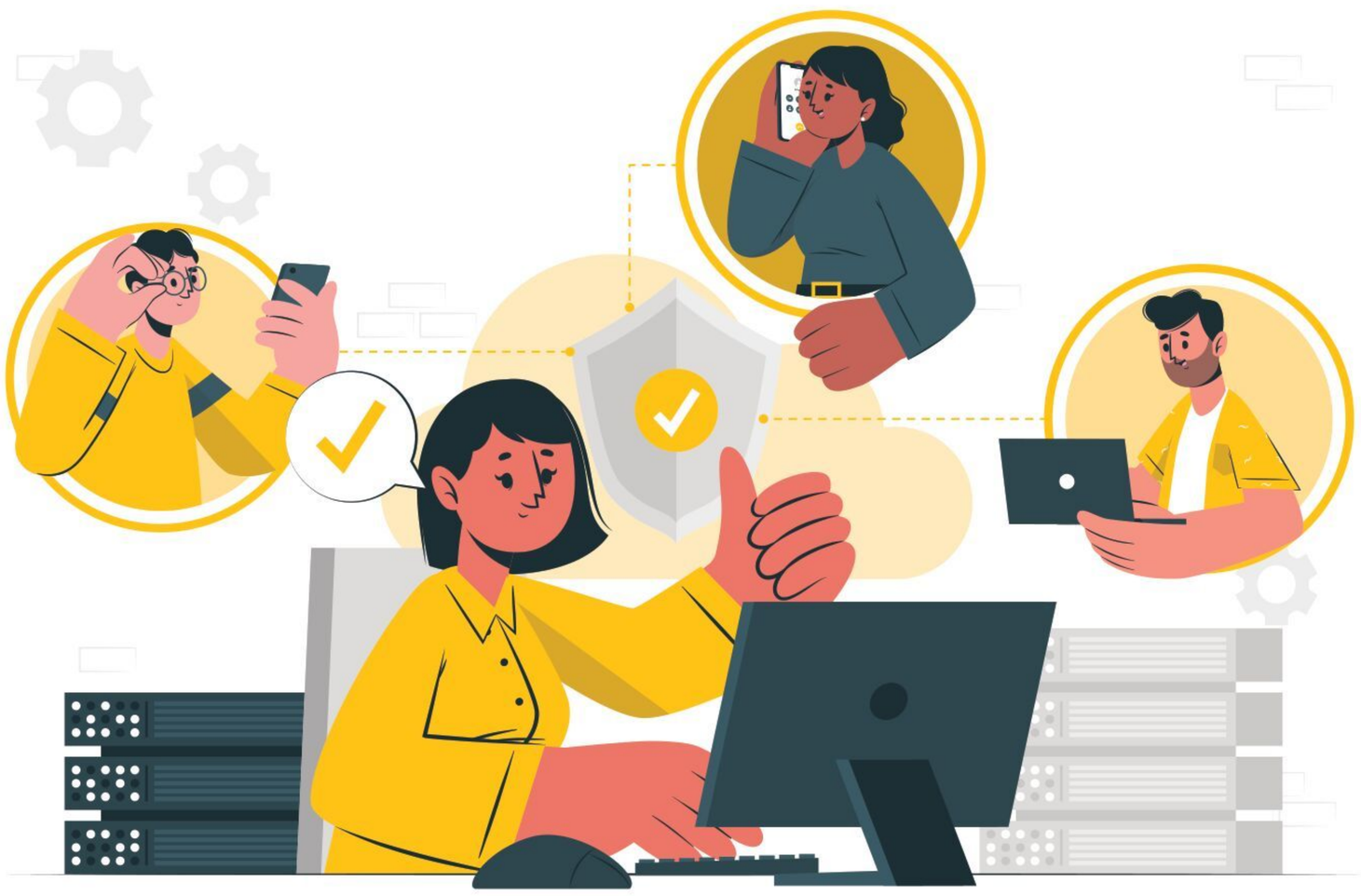
Subscription No. (for existing subscribers only)_____. **I would like to subscribe to the above (✓)marked magazine(s) starting with the next issue.** Please find enclosed a sum of Rs_____ by DD/MO/crossed cheque bearing the No._____ dt._____ in favour of EFY Enterprises Pvt Ltd, payable at Delhi.

Please mark one (nearest) relating to your subscription: ☐ **Indian Company** ☐ **MNC** ☐ **R&D organisation** ☐ **Engineering institute** ☐ **College/School** ☐ **Any other (specify):**_____

Send this filled-in form or copy to : EFY Enterprises Pvt Ltd, D-87/1 Okhla Industrial Area, Phase 1, New Delhi 110 020 | Ph: 011-40596600 | e-mail: support@efy.in

Terms:- # These rates are applicable for new subscribers as well as renewal by existing subscribers # Please allow 4-6 weeks for processing of your subscription.
Please include your pincode for prompt delivery of your copy.

Exploring Behavioural Patterns for **Asynchronous Communication**



A system delivers better throughput when various parts of it work fairly independently without blocking each other. Such a requirement demands asynchronous communication between consumers and providers. This third part of the 'The Design Odyssey' series explores a few of the GoF (Gang of Four) behavioural patterns that help in designing asynchronous communication at the object level. These patterns lay the foundation for popular messaging products like Apache Kafka and many others.

A consumer communicates its request to the provider and the provider, in turn, communicates the response to the consumer. Communication takes place in different forms.

Synchronous communication

In this mode of communication, the consumer sends a request to the provider and waits until it receives the response before proceeding further. In other words, the consumer gets blocked while the provider processes the request.

For example, in the following snippet, a consumer waits until the calculator returns the sum of two numbers.

```
calculator.add(10, 30);
```

Though there are valid cases for this mode of communication, as a side effect, it impacts the overall throughput of the system badly. What if the calculator takes an inordinate amount of time to respond? What if someone else is waiting for a response from the consumer object? As you can see, it can cascade further and the entire system may come to a standstill. For this reason, most modern systems avoid blocking communication between consumers and providers.

Callback

One of the time-tested mechanisms for establishing non-blocking communication between consumers and providers is the use of callbacks. The following snippet captures the essence of such a mechanism:

```
calculator.add(10, 30, sum -> {
     print(sum);
     stop(music);
});
play(music);
```

The objective of the above code is to play some music while the provider computes the sum of the numbers, and to stop the music once the results are printed. As a pattern, the consumer wants to do something (like playing music in this case) while the provider is busy processing the request. This is impossible in synchronous communication.

The third parameter in the *add()* call above is famously known as a *callback* function. A callback function is always executed sometime in the future after the original request is completed. The provider that offers such a mechanism immediately returns the control back to the consumer after receiving the request. The consumer is free to do whatever it

wants while the provider is busy with request processing. Once the provider completes processing the request, the callback function is executed. For all this to happen, the provider and the consumer run in separate threads or processes, and an underlying infrastructure handles the coordination.

Publish-Subscribe

The callback mechanism has a severe limitation, in its simplest form. It can handle only one-to-one communication between the provider and consumer. The consumers should be aware of who is the provider to make the request. When several providers and several consumers are involved, implementing a callback mechanism leads to the notorious *callback hell.*

A better alternative is to make the providers and consumers completely independent of each other. In other words, let the consumer only publish its request on a designated channel, instead of reaching out to a specific provider. On the other side, let the providers subscribe to the channel, process the requests, and publish the responses on a designated channel. The consumers are expected to subscribe to the responses from that channel.

The beauty of this mechanism lies in the fact that the providers and consumers are not even aware of each other. They only know the channels, not the publishers and subscribers. And, no one is blocking others. There can be any number of publishers and any number of subscribers. Moreover, there is no limit on the number of channels, number of requests, and number of responses. Since the requests and responses are decoupled, they are referred to as just messages. A message may carry a request, another message may carry a notification and some other message may carry a response.

The infrastructure that manages the channels, subscriptions, message delivery, etc, is the key in this mechanism. It is modelled along the lines of the following three behavioural patterns.

Mediator: In this pattern, a mediator acts as the infrastructure that connects the publishers with the subscribers. Several subscribers subscribe with the mediator for receiving the messages. Several publishers publish messages to the mediator. The mediator delivers the message to the subscribers. The mediator can also be equipped with some processing logic in terms of filtering the messages, routing the messages to a specific group of subscribers, etc. In some cases, a domain-specific mediator may also have business logic. Usually, an entity may act both as a publisher as well as a subscriber for different kinds of messages.

Observer: In this pattern, the subscribers register with a subject for receiving messages from specific sources. Often the sources are referred to as observables, messages are referred to as events, and subscribers are referred to as observers, in this pattern. The subject keeps the observers informed of the events from the observables.

In both the above patterns, the middle layer (mediator or subject) acts as the infrastructure through which the events or messages are delivered from the publishers or observables to the subscribers or observers.

Command: Though not specifically designed for this purpose, the command is essentially a callback function in the form of an object. In both mediator and observer patterns, it is the command object that processes a message or an event. The command object and its handling function are expected to be self-sufficient in such a way that they have all the information required for processing.

Messaging infrastructure

Having seen the basics of messaging patterns, it's time to build infrastructure for enabling asynchronous communication among the objects. Such an infrastructure is application-agnostic. Any application that requires asynchronous communication should be able to use our messaging infrastructure.

Here are the simple requirements:

- Objects should be able to register with the infrastructure for receiving messages from anyone.
- Objects should be able to register only for certain types of messages.
- Objects should be able to publish messages.

Implementation

Let's start with an interface that offers functions to subscribe, unsubscribe and publish:

```
package com.glarimy.framework;

public interface Broker {
     public long subscribe(Subscriber subscriber);
     public void unsubscribe(long sid);
     public void publish(String message, String type);
}
```

The idea is that once a subscriber is registered with the *Broker*, it returns a unique ID that can be used for unsubscribing in the future. Also, the *publish()* method accepts both the message as well as the type of the message.

The *Subscriber* is an interface with only one method that is meant for handling the received message.

```
package com.glarimy.framework;

public interface Subscriber {
     public void on(String message, String type);
}
```

Different applications may have different implementations for the Subscriber interface. Since the type of the message is

also passed as a parameter, the *on()* method can have a switch statement to deal with messages based on the type.

The simplest possible implementation of the *Broker* can store all the registered subscribers in memory.

```
package com.glarimy.framework;

import java.util.HashMap;
import java.util.Map;

public class MessageBroker implements Broker {
     private Map<Long, Subscriber> subscribers;

     public MessageBroker() {
       subscribers = new HashMap<Long, Subscriber>();
     }
     ...
}
```

The *subscribe()* method generates a unique ID before storing the subscriber in its list. A rudimentary implementation can use the current time as the ID, though it is not the best option for obvious reasons.

```
@Override
public long subscribe(Subscriber subscriber) {
     long sid = new Date().getTime();
     subscribers.put(sid, subscriber);
     return sid;
}
```

The *unsubscribe()* method requires no explanation.

```
@Override
public void unsubscribe(long sid) {
     subscribers.remove(sid);
}
```

And the *publish()* method iterates through all the subscribers and calls the *on()* method on each one of them.

```
@Override
public void publish(String message, String type) {
     subscribers.values().stream().forEach(s -> s.on(message,
type));
}
```

Note that this implementation does not give the non-blocking flavour of asynchronous communication. It only decouples the consumers and the providers. In order for real non-blocking communication, the *publish()* method can make use of a thread pool and call the *on()* method in separate threads.

This setup is depicted in Figure 1.

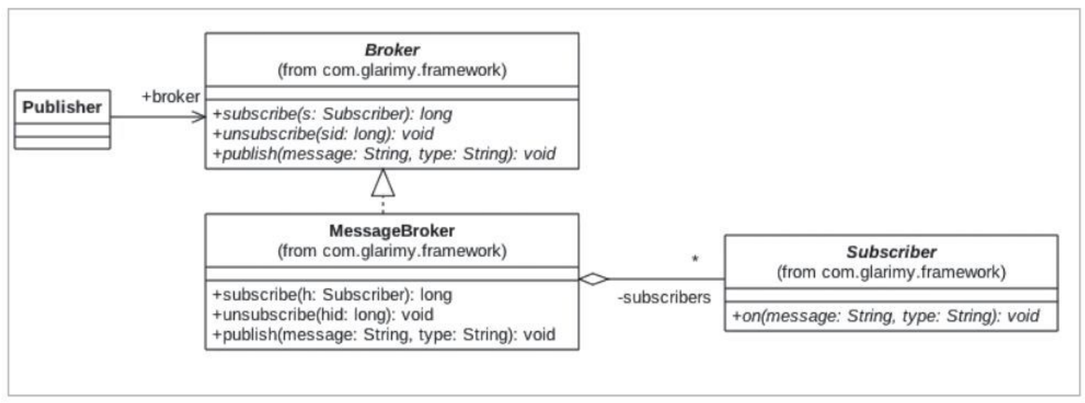


Figure 1: Messaging infrastructure

One semantic limitation of the above design is that a subscriber receives all the messages and uses a *switch* statement to handle them. This does not help us in improving the modularity of the application. When a new type of message is to be handled, the *on()* method of the subscriber needs to be modified with an additional case in the switch. Similar modifications are expected also when a type of message is to be ignored or handled differently.

Instead, an interface like *Handler* in the following snippet can be designed only to handle one type of message.

```
package com.glarimy.framework;

public interface Handler {
     public void handle(String message);
}
```

As can be observed, the handler just receives a message that can be handled without a switch statement. In other words, a given application can have different handler implementations for different types of messages.

Accordingly, the broker is refactored to register handlers, instead of subscribers.

```
package com.glarimy.framework;

public interface Broker {
     public long register(Handler handler);
     public void unregister(long hid);
     public void publish(String message, String type);
}
```

The broker implementation maintains a list of handlers and delivers the messages only to the matching handlers based on the type of message published. In order to identify the type of message that a handler is interested in, the interface can be updated as follows:

```
package com.glarimy.framework;

public interface Handler {
```

```
    public void handle(String message);
    public String getType();
}
```

The refactored implementation of the broker maintains the list of handlers.

```
package com.glarimy.framework;

import java.util.Date;
import java.util.HashMap;
import java.util.Map;

public class MessageBroker implements Broker {
    private Map<Long, Handler> handlers;

    public MessageBroker() {
      handlers = new HashMap<Long, Handler>();
    }
    ...
}
```

And the registration and removal of the handlers are on the expected lines.

```
@Override
public long register(Handler handler) {
    long hid = new Date().getTime();
    handlers.put(hid, handler);
    return hid;
}

@Override
public void unregister(long sid) {
    handlers.remove(sid);
}
```

However, the *publish()* method requires small modifications. Earlier, all the messages were delivered to all the subscribers. That has to be changed. It must only deliver messages to the handlers that are interested in the type of the message. Accordingly, the following is the updated *publish()* method.

```
@Override
public void publish(String message, String type) {
    handlers.values().stream()
          .filter(h -> h.getType().equalsIgnoreCase(type))
          .forEach(s -> s.handle(message));
}
```

Figure 2 depicts the updated broker design.

We will revisit the design of the message broker

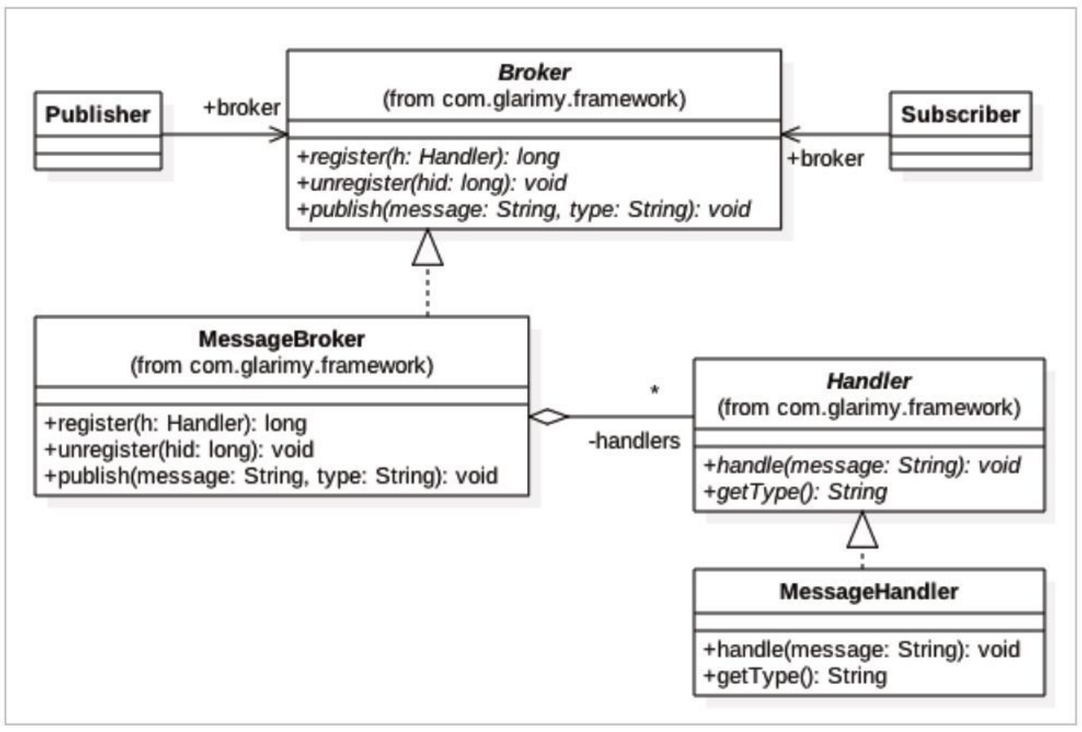


Figure 2: Messaging infrastructure with handlers

in the future for a deeper treatment. Since our current focus is on the fundamental object-oriented patterns, the *MessageBroker* in the current form is just enough to demonstrate the behavioural patterns.

UMS with messaging

The existing user management system (UMS) developed so far is intrinsically synchronous in nature. The consumers and providers are aware of each other. With the *MessageBroker* at its disposal, the UMS can be upgraded to use asynchronous communication.

Instead of the *UserRepositoryJournalProxy* directly calling the *JournalAdapter*, we can decouple them by using the broker. Let the *UserRepositoryJournalProxy* send messages to the broker and thereby the *JournalHandler* processes the messages. To connect these two virtually, the messages can be typed as *com.glarimy.ums*.

Following is the implementation of *JournalHandler*:

```
package com.glarimy.ums;

import com.glarimy.framework.Factory;
import com.glarimy.framework.Handler;
import com.glarimy.framework.ObjectFactory;

public class JournalHandler implements Handler {
    private JournalAdapter adapter;

    public JournalHandler() throws Exception {
      Factory factory = new ObjectFactory("config.
properties");
      this.adapter = (JournalAdapter) factory.
get("adapter");
    }

    @Override
    public void handle(String message) {
```

```
        adapter.record(message);
    }

    @Override
    public String getType() {
        return “com.glarimy.ums”;
    }
}
```

The corresponding implementation of *UserRepositoryJournalProxy* is:

```
package com.glarimy.ums;

import com.glarimy.framework.Broker;
import com.glarimy.framework.Factory;
import com.glarimy.framework.ObjectFactory;

public class UserRepositoryJournalProxy implements
UserRepository {
    private UserRepository target;
    private Broker broker;
    private static final String TOPIC = “com.glarimy.ums”;

    public UserRepositoryJournalProxy(UserRepository target)
throws Exception {
        this.target = target;
        Factory factory = new ObjectFactory(“config.
properties”);
        this.broker = (Broker) factory.get(“broker”);
        this.broker.register(new JournalHandler());
    }
    ...
}
```

Being a proxy, the *UserRepositoryJournalProxy* publishes messages to the broker as part of its pre- and post-processing routines.

```
@Override
public void add(Long phone, String name) {
    broker.publish(“c=repository, s=add, a=” + “&” + name,
TOPIC);
    target.add(phone, name);
    broker.publish(“c=repository, s=add, r=void”, TOPIC);
}

@Override
public String find(Long phone) {
    broker.publish(“c=repository, s=find, a=” + phone,
TOPIC);
    String name = target.find(phone);
    broker.publish(“c=repository, s=find, r=” + name, TOPIC);
    return name;
}
```

The updated design of the UMS is depicted in Figure 3.

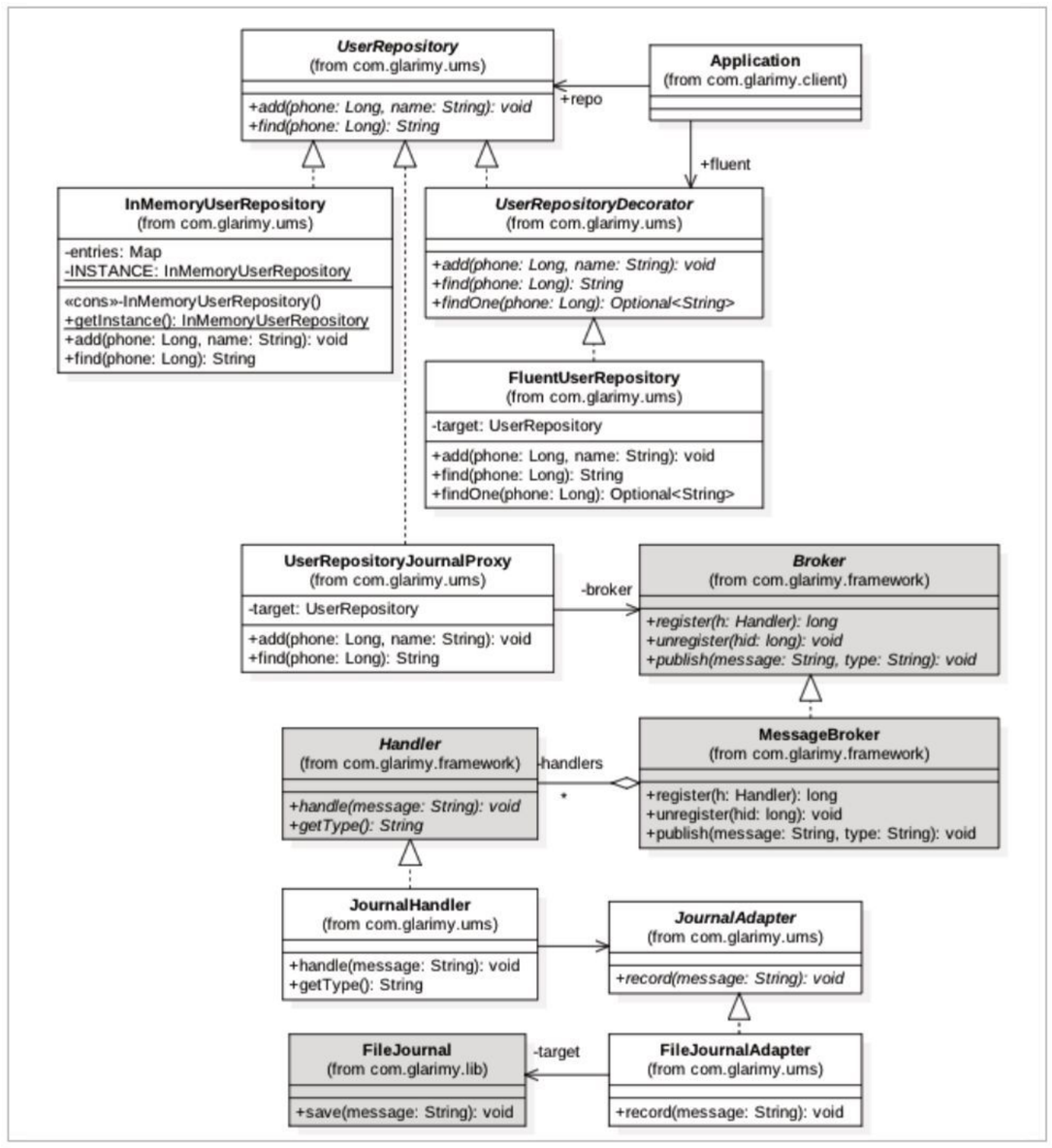


Figure 3: UMS with messaging

An important observation is that the *UserRepositoryJournalProxy* and the *JournalAdapter* are now completely decoupled. In fact, in response to the messages published by the *UserRepositoryJournalProxy*, any number of handlers on the topic *com.glarimy.ums* can respond by their own processing. For example, a handler may send an email and another handler may send an SMS confirming the addition of a user. Registering or unregistering such handlers does not have any bearing on the rest of the system.

If the broker invokes the handlers in separate threads, this implementation realises a complete non-blocking asynchronous communication.

With this, we leave the object-oriented design using GoF patterns. In the next part, we will explore domain-driven design. END

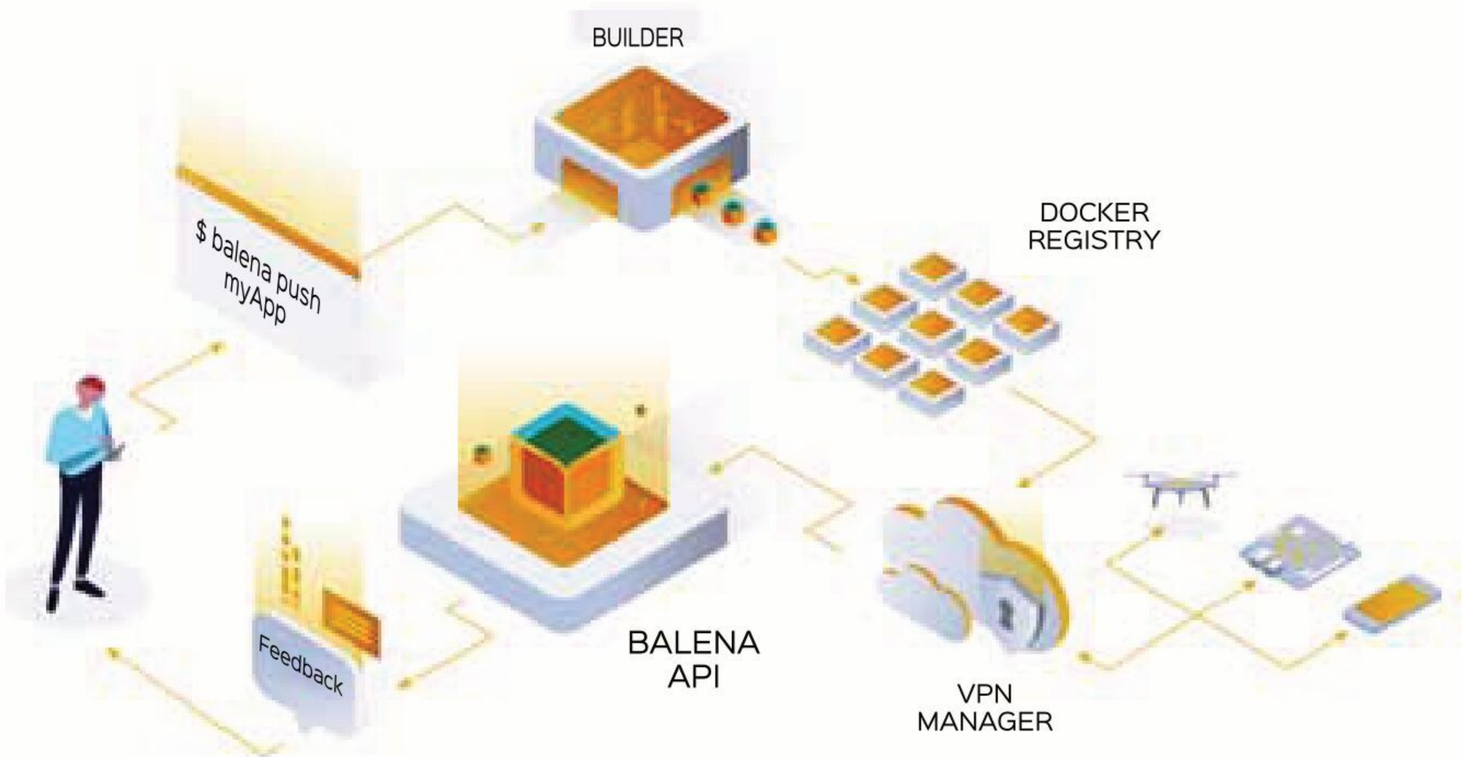
By: Krishna Mohan Koyya

The author is the founder and the principal technology consultant at Glarimy Technology Services, Bengaluru. He has delivered more than 500 programs to corporate clients on design and architecture in the last 12 years. Prior to that, he held various positions in the industry and academia for a decade.

FOCUS

Connecting Devices

Balena: Helping to Manage IoT Fleets



IoT fleet owners who use numerous connected devices can manage and monitor these with the help of the balena ecosystem. This article outlines how this can be done. *Read more on page.....24*

Using Node-RED for IoT and Edge Computing

All of us make use of IoT in one way or the other today, and this usage will only grow in the near future. This article details how Node-RED, a powerful and easy-to-use programming platform, helps in implementing IoT and edge computing.

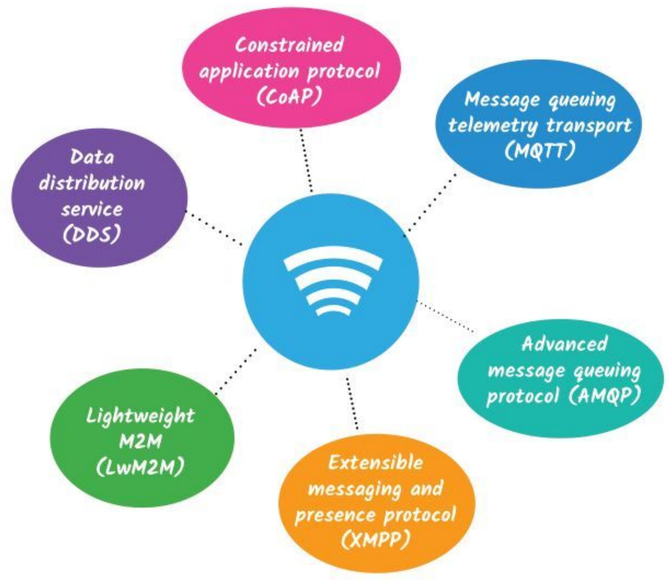
Read more on page.....44

Keep the Fish Healthy with this IoT Based Aquarium

Fish living in aquariums need constant monitoring. This article outlines the development of an IoT based monitoring system for aquariums, which eliminates the need for human intervention to check the water quality in them.

Read more on page.....32

Choose the Most Optimal IoT Protocol for Your Project



The Internet of Things (IoT) is a term used to describe the interconnectivity and intercommunication of everyday devices with the Internet. Everything, from a smartwatch to a smart light bulb, is part of IoT. We also discuss which IoT protocol might be ideal for your project. If you are an IoT enthusiast looking to get started with your own project, this article is a must read.

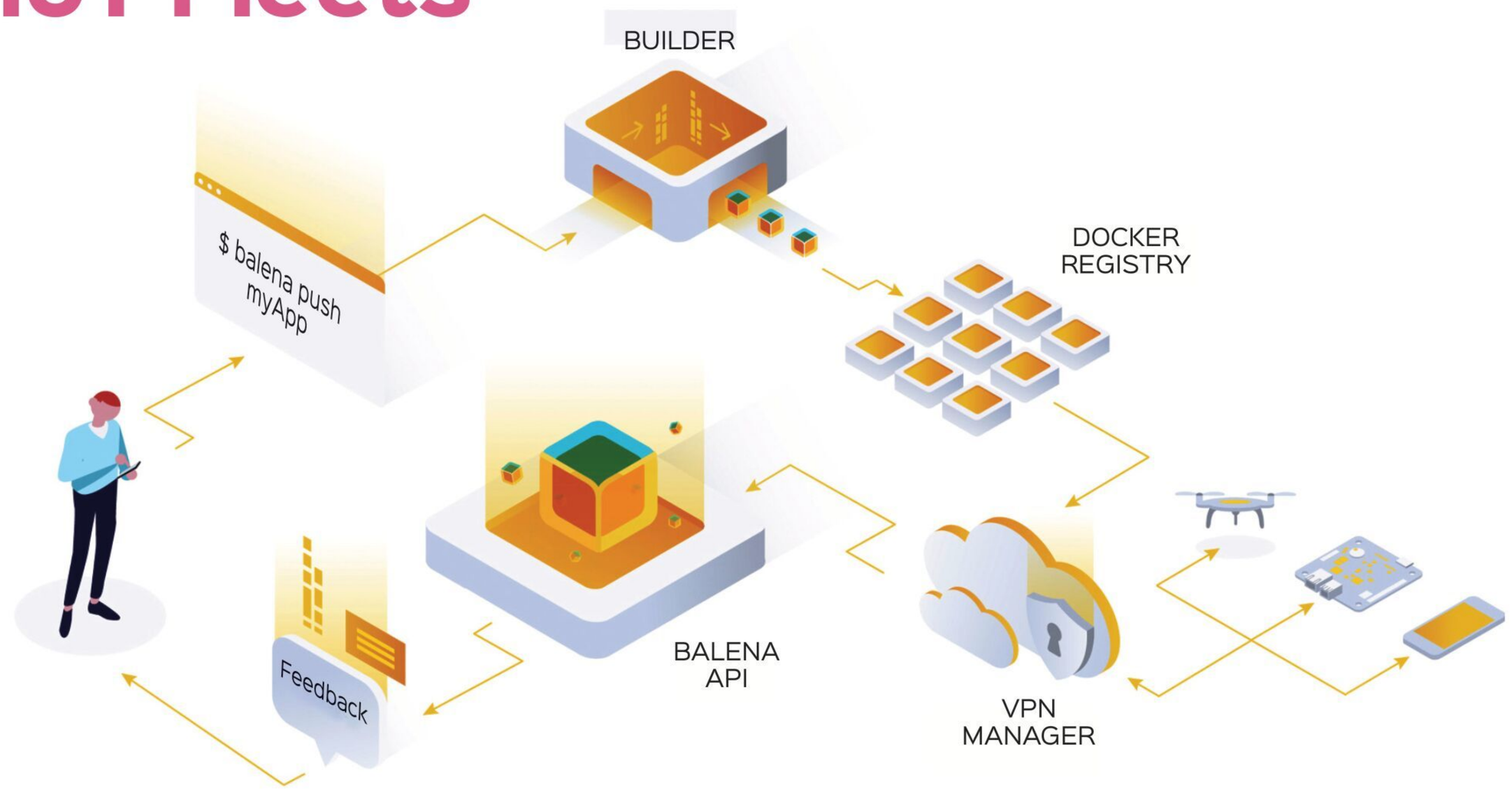
Read more on page.....39

Installing and Using Node-RED on Ubuntu

Node-RED adds to the benefits of Node.js. This article shows how to install and use it on the popular open source operating system Ubuntu.

Read more on page.....53

Balena: Helping to Manage IoT Fleets



IoT fleet owners who use numerous connected devices can manage and monitor these with the help of the balena ecosystem. This article outlines how this can be done.

The Internet of Things (IoT) is making the lives of millions of people better every day, whether it is the ease of monitoring your home from the far corners of the world, fitness bands sensing step counts and syncing them with your smartphones, GPS trackers installed inside each truck in a logistics company's fleet giving updates on package movement, or thousands of tiny sensors distributed over a big farm that continuously report soil health for better crop yields.

IoT has become a subtle part of our lives, but like all other good things, there are challenges to keeping the overall experience natural, seamless and secure.

From a manufacturer or IoT fleet owner's point-of-view, there are certain challenges that need to be addressed to ensure the end user experience is smooth and secure. This includes providing regular software updates on these devices so that all the security vulnerabilities are patched and new features are continuously rolled out.

Many applications involve receiving data from these edge devices deployed in the field, making it extremely important to monitor them remotely for their health status (battery, network, storage), and update them as and when needed. The last thing a fleet owner would want is to send someone to manually update or fix an IoT device which is deployed out in the wild.

Addressing challenges in IoT fleet management

IoT fleet owners need to build a complete infrastructure to remotely monitor and securely update edge devices. They also need the resources to develop and keep this infrastructure up and running. This can become a pain point when you're managing heterogeneous devices with different architectures (say, x86 and ARM) and different network connectivity types (Ethernet/Wi-Fi), and when you need to add more devices every now and then. As you grow your fleet, you will have thousands of devices and your infrastructure will need to be scaled accordingly.

Setting up this infrastructure in-house takes your focus away from your main application and the problem you are trying to solve as a fleet owner.

What is balena?

Thankfully, there are solutions like balena.io, which is a collection of tools to address the challenges IoT fleet owners

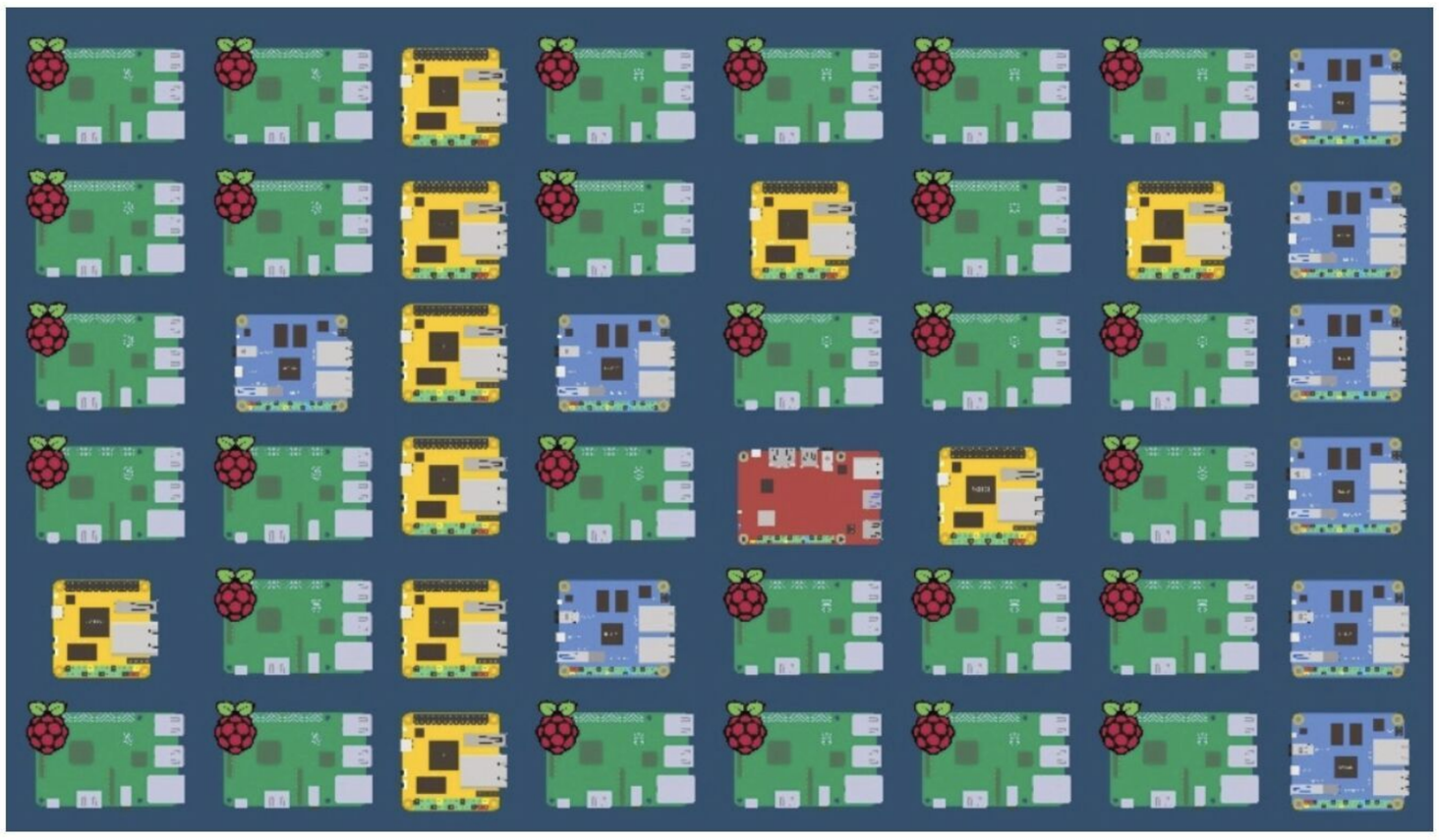


Figure 1: Scaling your IoT fleet

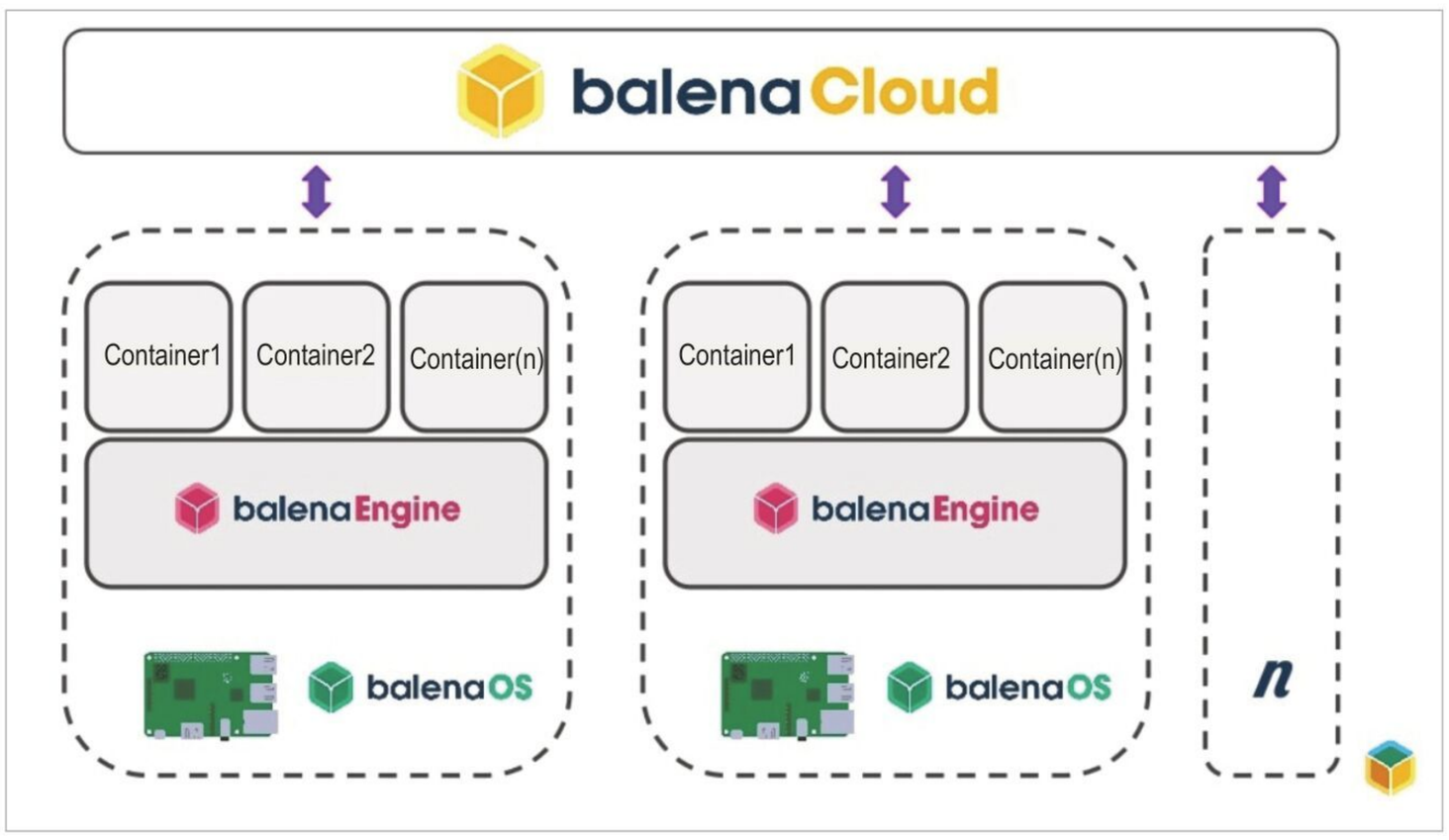


Figure 2: balena ecosystem

face. The balena ecosystem allows them to manage multiple edge devices straight from a single dashboard (balenaCloud), and adding a new device to an existing fleet is extremely straightforward. So you can focus on your main application rather than worrying about developing and scaling the infrastructure.

The balena ecosystem provides various tools that enable edge developers to adopt a container based Web-like development workflow, in which your application and supporting pieces reside inside Docker containers, managed by balenaEngine. The latter is similar to dockerEngine but suitably adapted for edge development. It runs on top of balenaOS, a Yocto based rugged embedded Linux operating system meant for single board computers supporting ARM32, ARM64 and x86. A special container called supervisor, which comes baked into balenaOS, manages all the connection, provisioning and updation work from the device itself. A fleet owner can manage all the devices, add more and roll out an update release from balenaCloud, which provides a dashboard to monitor and manage all your devices in the fleet.

The balena ecosystem

Before diving into how you can get started with deploying your own fleet using balena, here's a quick recap of a few essential utilities.

- *balenaOS*: A Yocto based Linux operating system meant for edge devices; for example, Raspberry Pi
- *balenaEngine:* Container manager built into balenaOS, adapted for edge devices
- *balenaCloud:* Dashboard to monitor, manage, add and update devices in your fleet
- *balenaCLI*: A command line utility to push releases of your project to balenaCloud. balena also supports Git for this purpose, but for this tutorial we'll be using balenaCLI
- *balenaEtcher:* Some of you may already be using this great tool by balena for flashing SD cards/ USB drives

It is a good time to mention that most (if not all) of these tools are open source and free to use; they are hosted on the balena GitHub account.

Note: You can add up to 10 edge devices for free, after which you need to buy the balenaCloud subscription.

Deploying your first IoT fleet with balena

We will now deploy our first IoT fleet with a sample 'hello-world' application. Here are the steps to do so.

Step 1: Create an account on balenaCloud
Step 2: Create a new fleet
Step 3: Add your device(s)
Step 4: Push your code to the devices in the fleet

Create an account: Visit balenaCloud (*https://www.balena.io/cloud/)* and create a free account. You can also log in using your Google or GitHub account.

Download balenaCLI: Download balenaCLI for your operating system from *https://github.com/balena-io/balena-cli/.*

Flash the SD card: Download balenaEtcher for your operating system to flash the SD card (from *https://www.balena.io/etcher/*,) or you can use *build*

An EFY GROUP EVENT

5243 ATTENDEES

70+ SESSIONS HOSTED

100+ SPEAKERS

18 COUNTRIES REPRESENTED

18th Edition

OPEN SOURCE INDIA

VIRTUAL EVENT

Thank You

VISITORS, for making the event a huge success

SPEAKERS, for contributing your valuable thoughts

PARTNERS, for your support

Microsoft Azure	soda foundation	Google	aws
MySQL	HUAWEI	iTEXT	GitHub
SUSE	ENTREPRENEUR FIRST JOINEF.COM	TRUEFOUNDRY BY ENSEMBLE LABS	Application Library Engineering Group
openLooKeng	EDGE GALLERY	soda foundation meetupindia	BANGALORE CNCF MEETUP

www.OpenSourceIndia.in

For more details, call on +91-9667743666 or email us at info@opensourceindia.in

in tools like *dd* for Linux.

Clone hello-world project: Clone the balena Node.js hello-world example from the GitHub repository at *https://github.com/balena-io-examples/balena-node-hello-world* using Git clone.

Hardware needed

Since this is a hello-world project with the balena ecosystem, we'll keep it simple and use minimal hardware. We will be hosting a basic Web page created in Node.js on the Raspberry Pi, but we can always add sensors, actuators and displays as per our requirements. For this project, we'll need:

- A Raspberry Pi board
- SD card
- Power supply

Setting up the fleet

Step 1: **Create a new fleet**

The first step is to create a fleet on the balena cloud dashboard after logging in.

Here, you need to name your fleet; for this example, we'll call it hello-world. You can also choose a default device type for this fleet, which will be Raspberry Pi 4. We can name the fleet type *Starter* for now. Click on 'Create new fleet' to finish the initial setup. Once a fleet is created, you can add devices to the fleet using the 'Add device' button.

Step 2: **Add a new device to the fleet**

This page will allow us to download balenaOS for our device. Here, we can also select the device you want to add. Raspberry Pi 4 is here by default from the fleet creation step, but you can change it. Choose the recommended version of balenaOS.

We also have the option to choose the balenaOS image for production that is more secure. The locked in version is meant to be deployed in production with ssh disabled, but we'll choose a development image for this demo since we're not deploying this device in the field yet. The good thing is that you can also choose how you want to connect the device to the Internet. You can choose only Ethernet or Ethernet + Wi-Fi. You can also add your Wi-Fi credentials on this page, so that the balenaOS image you download here comes baked with these credentials and the device automatically connects with the hotspot. This is quite handy for quick prototyping.

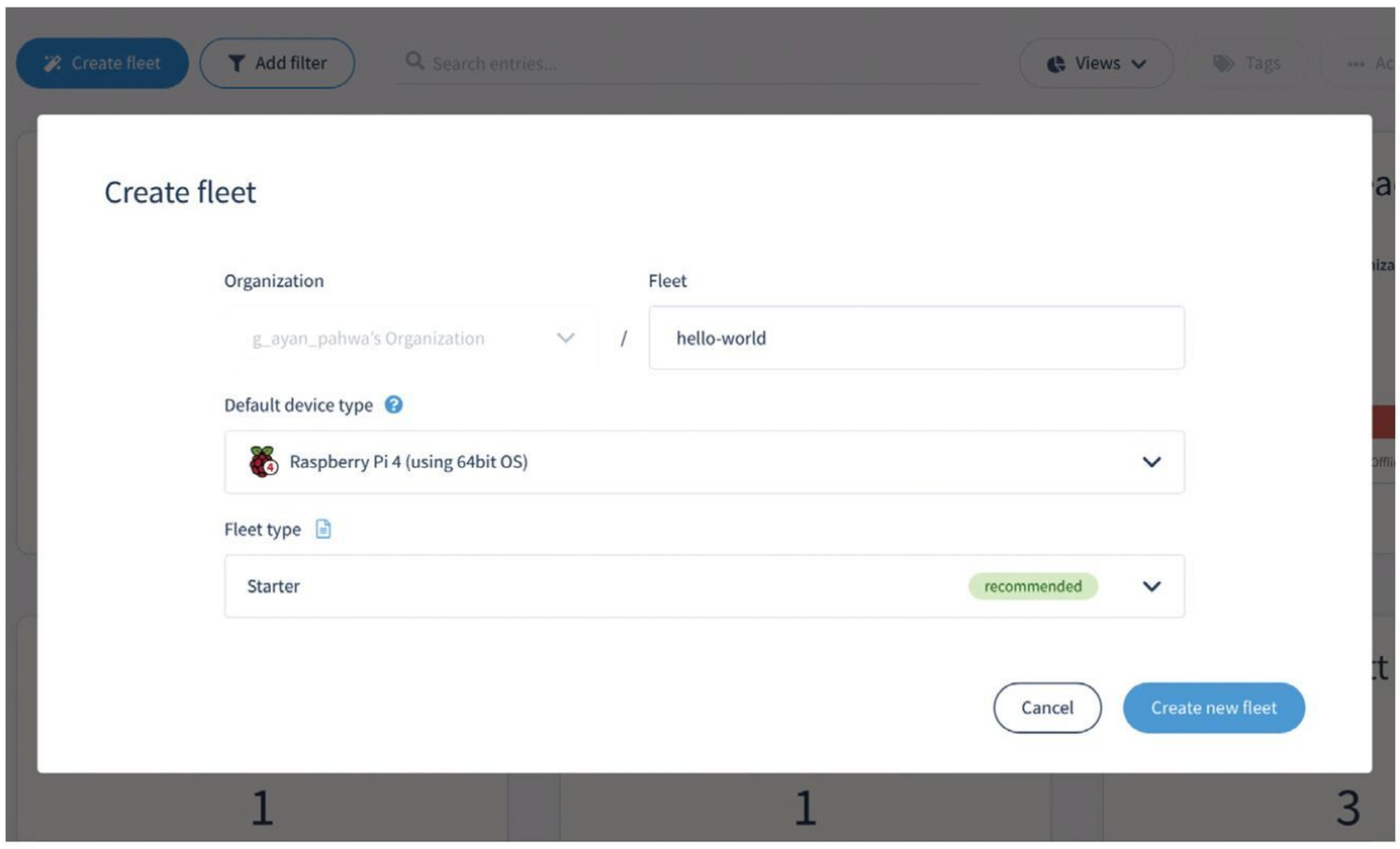


Figure 3: Creating a new fleet

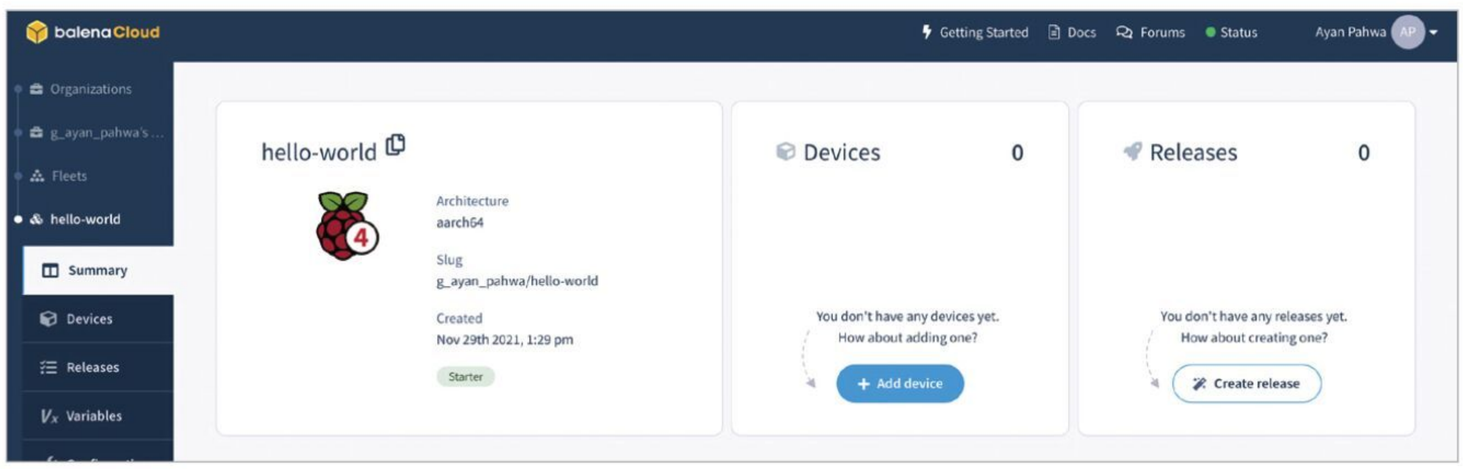


Figure 4: Fleet summary

Step 3: **Pushing code to the device(s)**

After powering on the Raspberry Pi, if you go to the device pane on the balenaCloud dashboard you'll be able to see your device online. We now need to push the code onto our device, for which we will use balenaCLI.

In the terminal (Linux/Mac) or balenaCLI.exe (Windows), execute the following command:

```
balena push <fleet name>
```

To associate your balenaCloud account with balenaCLI, you can choose any available login method. The preferred one is Web authorisation, which will authenticate using a Web browser. Finally, to push the code to the devices in the fleet, we'll use:

```
balena push hello-world
```

This will update the devices with the latest release.

How does this work?

Balena push checks for *dockerfile* or *docker-compose* file and *build* container images for your device architecture, and makes your balenaCloud dashboard available for updating all the devices in the fleet. The images are built on the balena infrastructure, but there's also a way to build images locally using something called local-mode.

Now let's verify if our app is working off the Raspberry Pi device. From the

www.opensourceforu.com

Job Guaranteed.

You don't need a training institute. Just do it yourself.

WORK WITH OPEN SOURCE.

Demand for Open Source is sky rocketing. Be it for managing IT infrastructure or development of software —Open Source experts are what organisations are seeking. Develop expertise in an Open Source stack, and watch your career graph zoom.

You don't need a training institute for this. You can do it yourself. And, *Open Source For You* can be your friend and a guide through this journey.

TO READ OUR PRINT EDITION Visit: https://subscribe.efyindia.com

TO READ OUR EZINE EDITION Visit: https://ezine.lfymag.com

WORLD'S LEADING PUBLICATION ON OPEN SOURCE

Looking for marketing solutions to engage with cutting edge techies?
Contact us at growmybiz@efy.in OR call us at +91-9811155335.

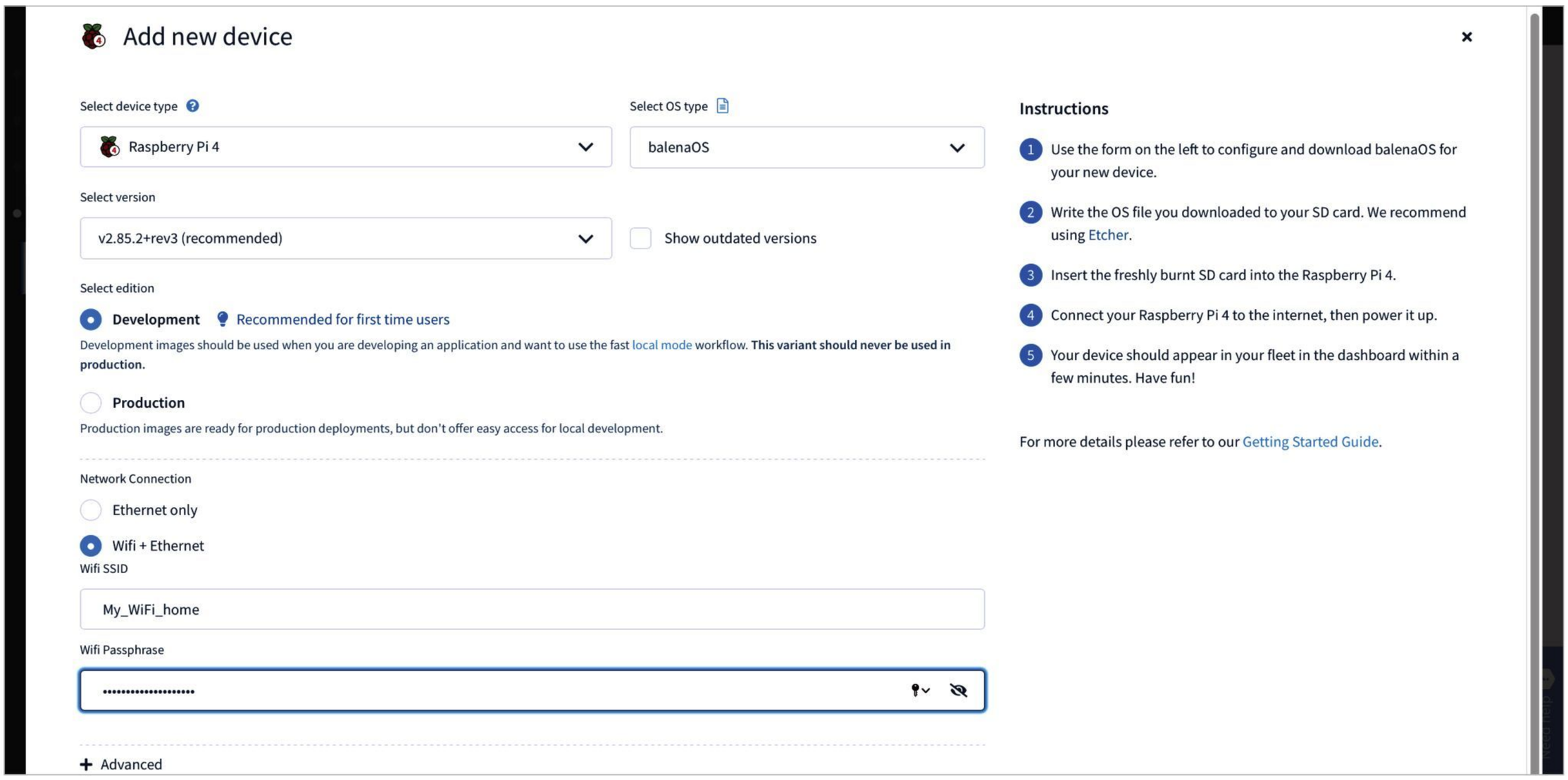


Figure 5: Add a new device

Figure 6: Output of successful balena push; the unicorn Charlie means build is successful and images are pushed to the fleet

balenaCloud dashboard, copy the local device IP address and open it in a new tab. Voila! You'll be greeted with a Web page hosted off from the Pi itself. The best part is that you can enable the 'Public Device' URL and access it from anywhere in the world. How handy is that!

Digging a little deeper

If you check the summary page on the balena dashboard, you will find a lot about your device and fleet. For example, in the service section, you can see there is only one service or one container running called *main*, but you can also run an application with multiple containers using docker-compose. As an example, an application running the Mosquitto MQTT broker, InfluxDB and Grafana can be achieved using a docker-compose file.

From the same dashboard, you can see the device's health like CPU utilisation, temperature, memory, and storage. You can also reboot, power-off the device, update balenaOS, supervisor, etc. There is a built-in log viewer,

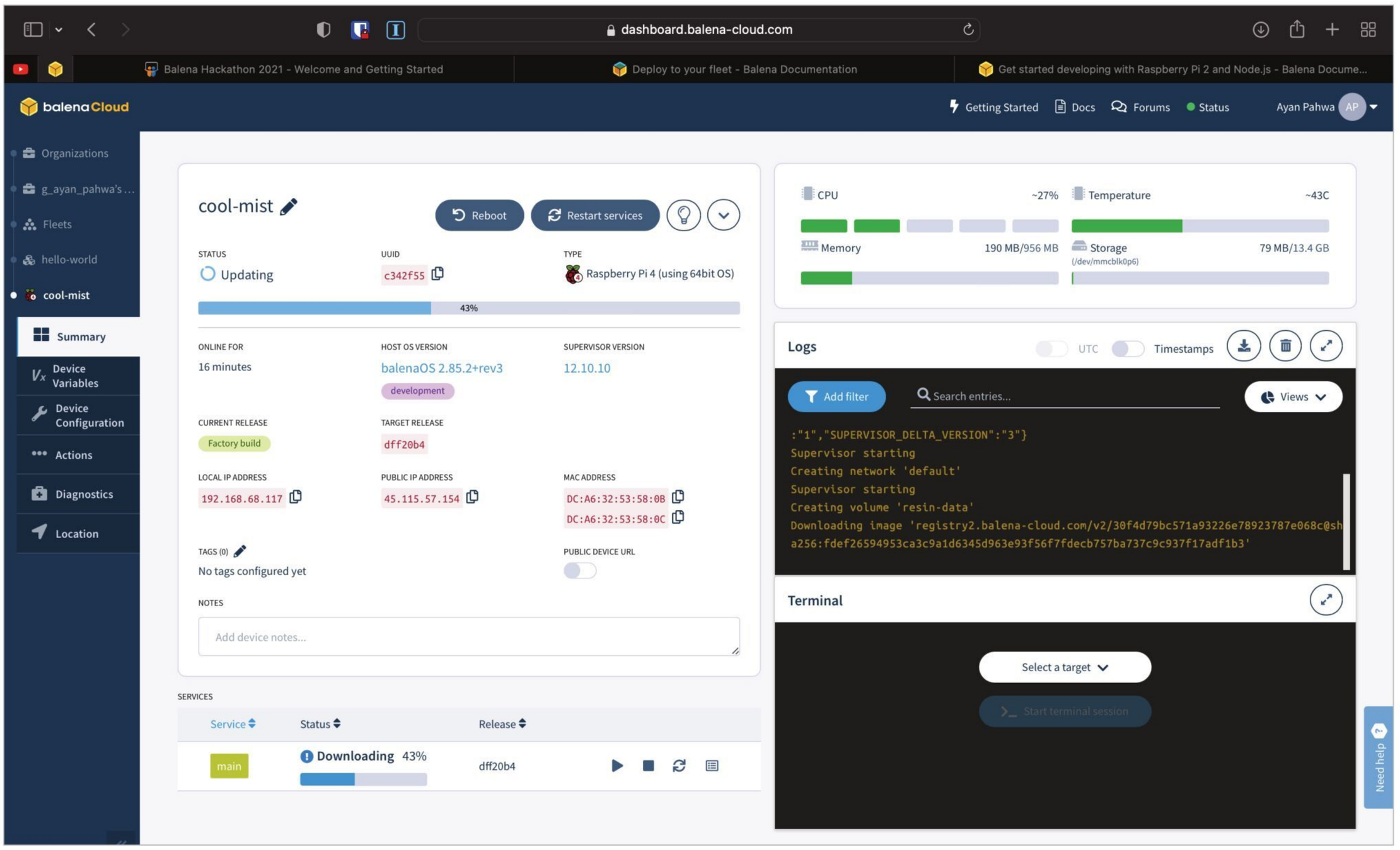


Figure 7: After a successful build, images are made available to devices for updation

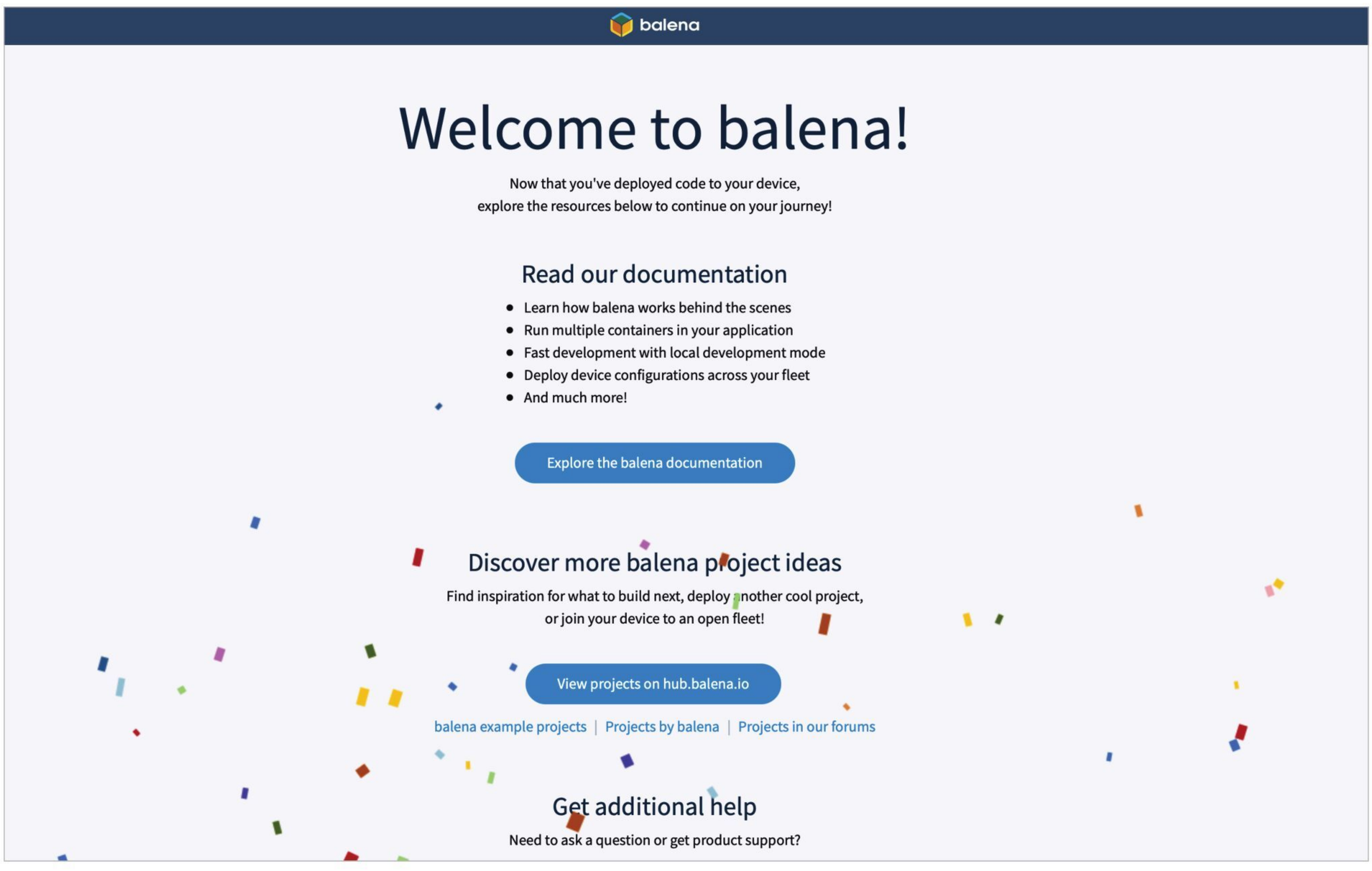


Figure 8: Page hosted from the Raspberry Pi

Continued to page no...38

Keep the Fish Healthy with this IoT Based Aquarium

Fish living in aquariums need constant monitoring. This article outlines the development of an IoT based monitoring system for aquariums, which eliminates the need for human intervention to check the water quality in them.

Beautiful and delicate marine life creatures require a lot of care. Misconceptions such as fish can survive without food or can live with low water quality in an aquarium can endanger them. Fish require sufficient levels of nutrients in the water they swim in. In fact, the oxygen content in the water is vital. Dissolved oxygen (DO) is a measure of the amount of oxygen present in the water. Most fish need DO content in the range of 5-6ppm in a water tank, even if some can manage at 4ppm. Anything lesser is dangerous. The pH value is also an important aspect in determining the quality of the water. This value needs to be monitored frequently to maintain a healthy environment for the fish in an aquarium.

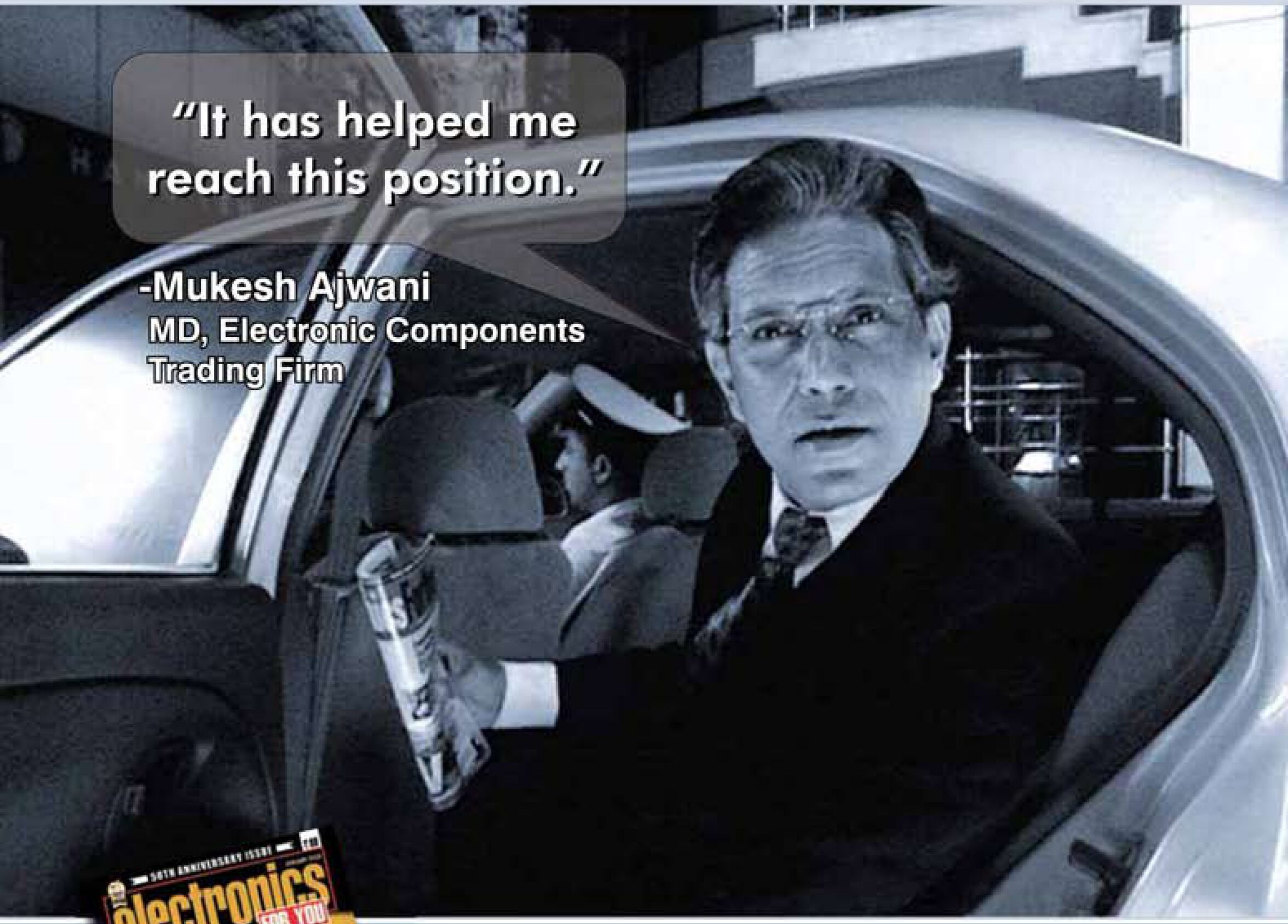
Let us now use the Internet of Things (IoT) to run a framework that can detect and observe oxygen levels, temperature, pH of the water, the water level, and so on, in an aquarium by using a few sensors. Keeping up with the right conditions in the water is extremely hard manually. Automation can significantly diminish the fatality rate and help make an aquarium a healthy environment for fish to live in.

Making an aquarium intelligent

The system we are developing aims to couple existing fish automation systems such as water-quality monitoring with automating the tank cleaning process, the oxygenator, the temperature filter, pH value sensor and the DO sensor. A temperature sensor is

PIONEERS IN TECHNOLOGY.
PARTNERS IN SUCCESS.

EFY GROUP
YOURS SINCE 1969



Mr Ajwani is not the only one who owes his success to *Electronics For You*. The technology magazine that started off in 1969 is today, South Asia's most popular one. Covering the latest on emerging technologies, changing industry trends, evolving trade practices...

Not surprisingly, top decision makers and those high up on the corporate ladder in the electronics fraternity treat EFY as their Bible.

- **Over half a million readers**
- **India's largest selling technology magazine**
- **Caters to the entire electronics fraternity**

D-87/1, Okhla Industrial Area, Phase 1, New Delhi-20 • Ph.: 011-26810601/02/03 • Email: info@efy.in • www.electronicsforu.com

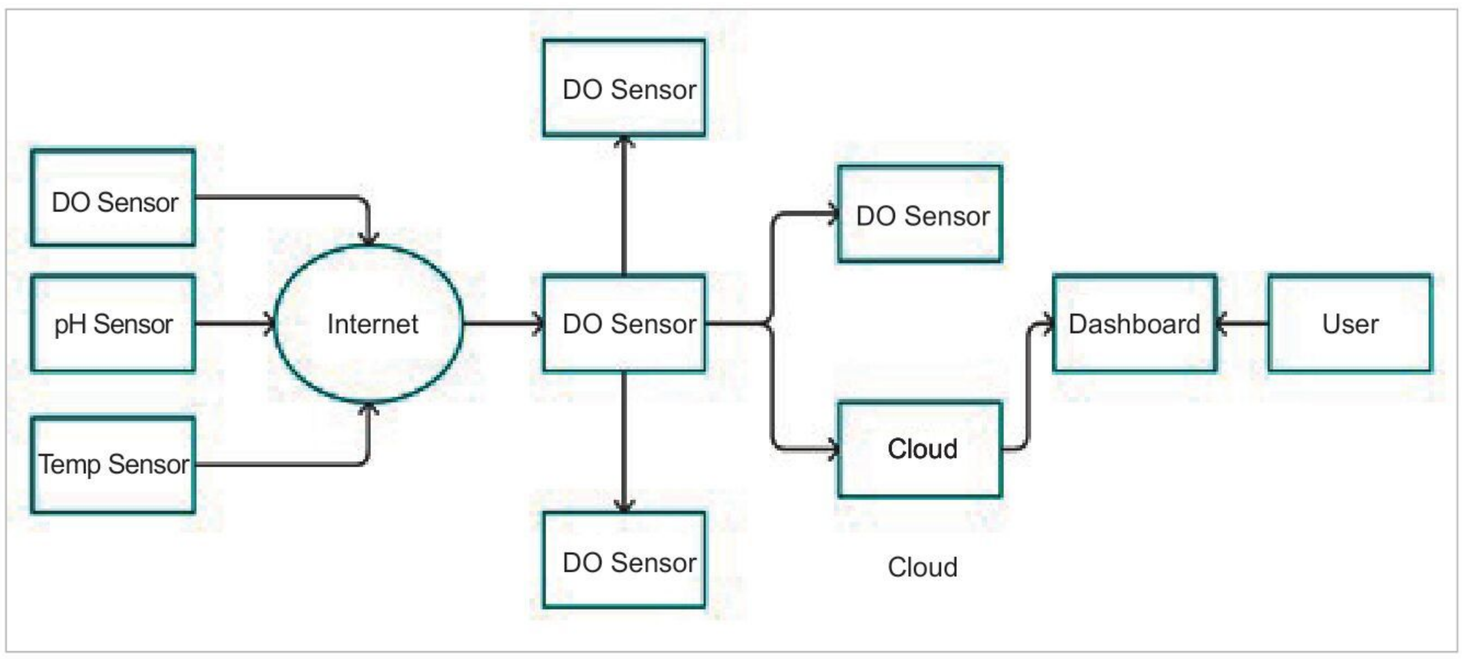


Figure 1: Block diagram of iAqaurium

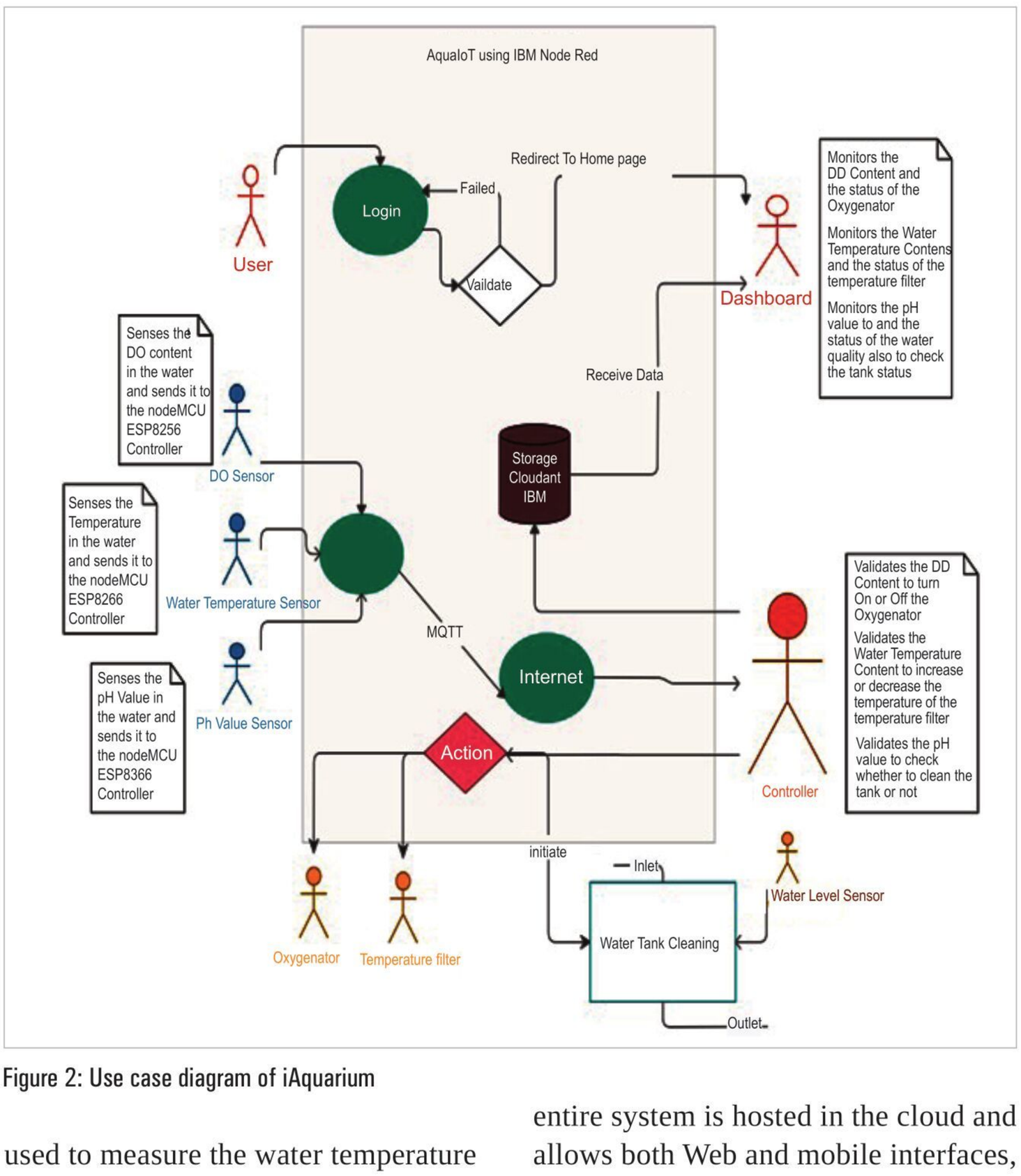


Figure 2: Use case diagram of iAquarium

used to measure the water temperature and data is used to control it. The normal and optimal water temperature for most fish lies between 21°C and 26°C. This automation system is designed with sensor modules (input data), sensor data action (output), cleaning of the tank, along with cloud integration and a Web interface. The entire system is hosted in the cloud and allows both Web and mobile interfaces, so the user can monitor it from any part of the world. It is simulated using the Node-RED environment with the IBM Cloudant database. We will call this system iAquarium.

iAquarium system design

Figure 1 represents the block diagram of the iAquarium system with all its functionalities. The use case diagram is depicted in Figure 2.

Steps to design the iAquarium

- Install the latest version of Node.js from *https://nodejs.org/en/download/*.
- Install Node-RED using the command *npm install –g node-red.*
- From the browser, get access to the Node-RED flow using the IP *http://127.0.0.1:1880.*
- From the Node palette of Node-RED, insert the appropriate nodes, configure their properties and wire them.
- Click the *Deploy* button and confirm deployment.
- Visualise the output through the *Debug* monitor window and dashboard UI.

Node flow design of iAquarium

1. Insert *inject* node to trigger the event.
2. Insert *function* node and write the appropriate JavaScript code for automation.
3. Insert *join* node and *MQTT* node to transmit alert messages through HiveMQ.
4. Insert *cloudant out* node to push data onto the cloud.
5. Insert *template* node to create a UI design.
6. Insert *debug* node for printing msg.payload.

iAquarium working modules

Sensor data: This module involves data collection from all the sensors in the system like pH sensor, DO sensor, and water temperature sensor. The sensor values are generated in Node-RED seamlessly at regular intervals of 3 seconds, as shown in Figure 3. The data is then sent to the controller, through the *MQTT out* node using the HiveMQ broker. The DO content is measured in ppm units and the water temperature in Celsius.

Controller: This represents the action taken based on the sensor data, as shown in Figure 4. Through the *function* node, appropriate conditions for the oxygenator are coded using JavaScript. The oxygenator is used to increase the oxygen content in the tank, as a result of the validation of the DO content in the water found by the DO sensor. If the content drops below 5, the oxygenator is turned on to increase it. The temperature filter heats or cools the water temperature to maintain the tropical temperature in the tank as a result of the information sent to the controller. If the temperature drops or increases due to the environment around, the filter regulates and maintains the desired temperature.

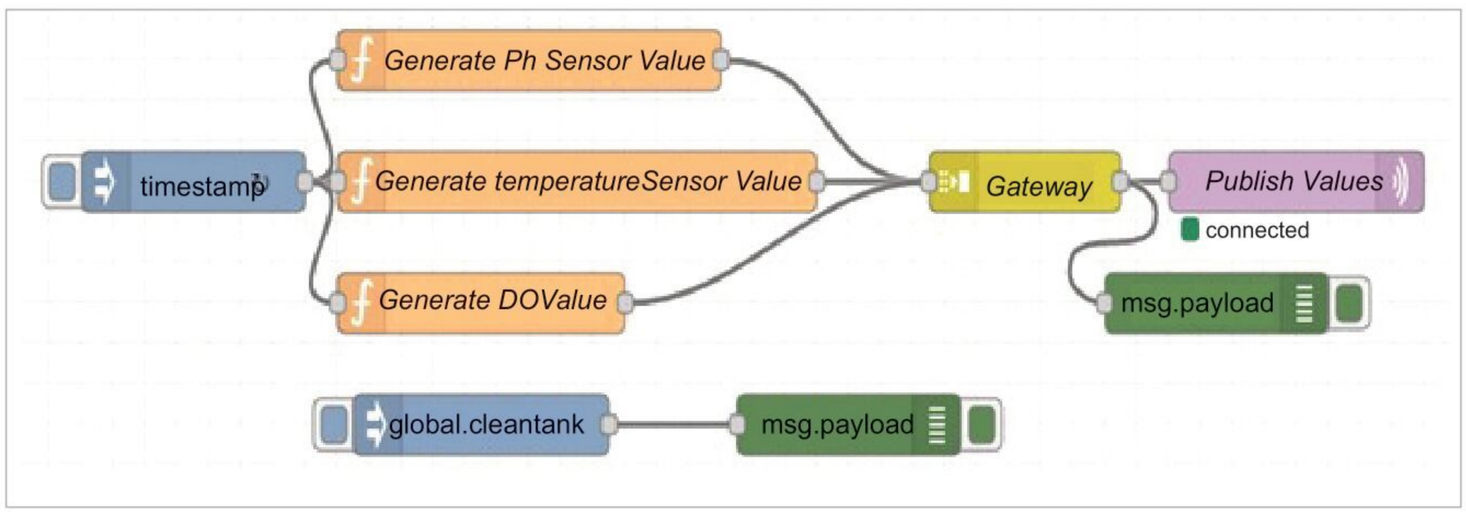


Figure 3: Sensor data generation through Node-RED

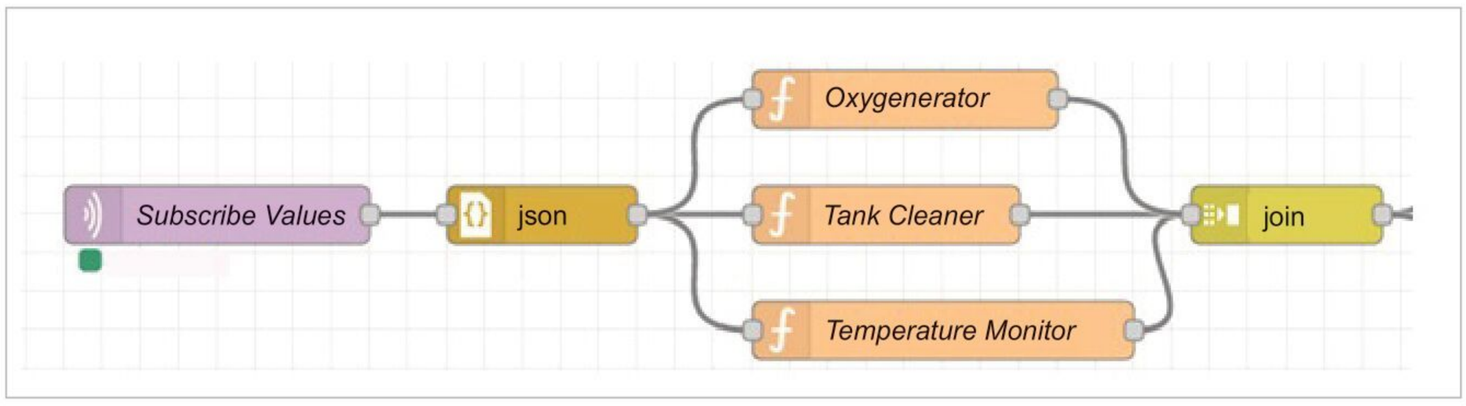


Figure 4: Oxygenator using Node-RED

Pseudocode for oxygenator (to be written in *function* node using JavaScript coding)

```
Initialize oxygenate =0.25
Check for tank clean status
If DO <5 then add DO +oxygenate
Send status as “Low DO level, turn on
Oxygenate”
Else send status as “Normal DO level”
```

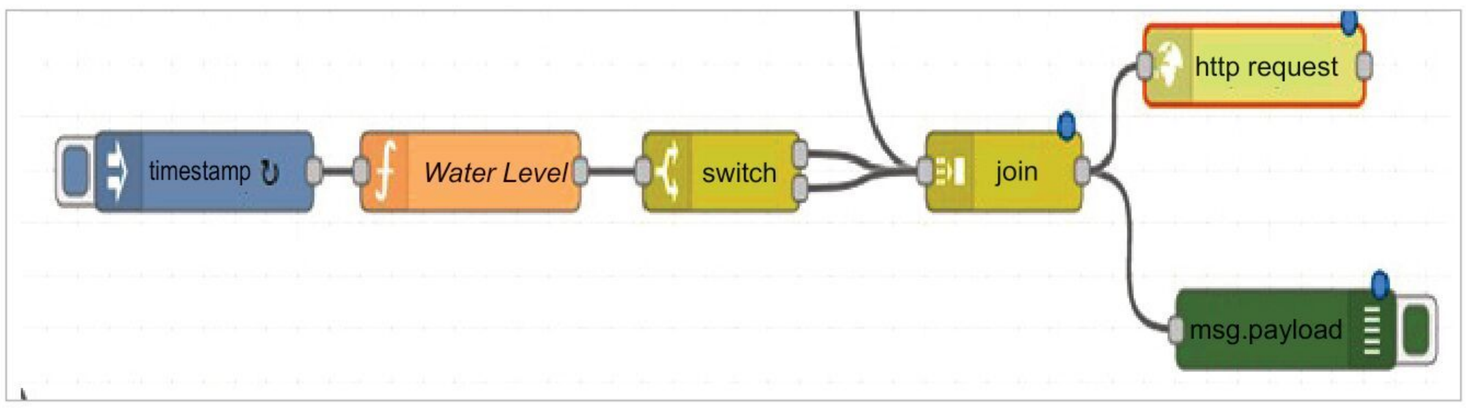


Figure 5: pH monitor and tank cleaner controller by Node-RED

Tank cleaning: The tank cleaning process is an action simulated as a result of the validation of the pH value sent to the controller by the pH sensor, as seen in Figure 5. Through the *function* node, appropriate conditions for the pH value are coded using JavaScript. If the pH value drops below 6.4 or increases above 7.6, it indicates either more acidic or basic content in the water, and the cleaning process is initiated. The system then alerts other sensor modules to halt. This module has a water level indicator to find the level of the water in the tank, a water inlet, and an outlet. When the process starts, the water outlet is opened and 70 per cent of the water is let out. Then the water inlet is turned on to fill the water in the tank.

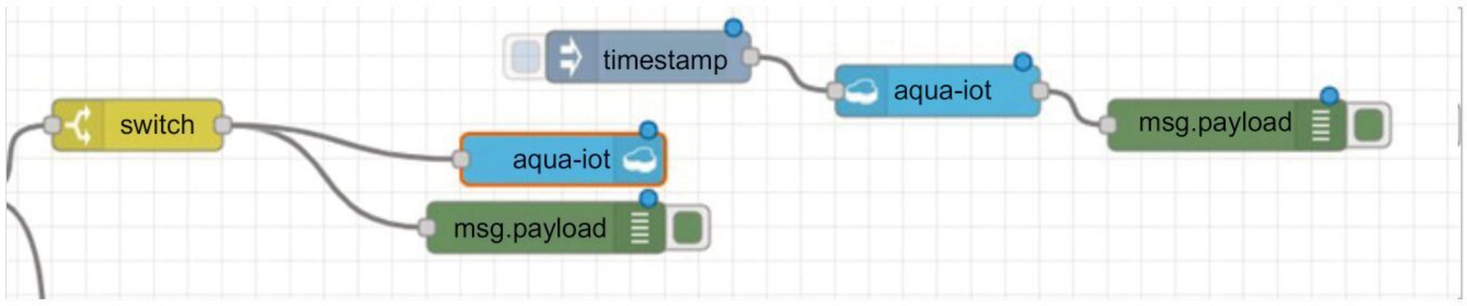


Figure 6: *Cloudant out* in Node-RED

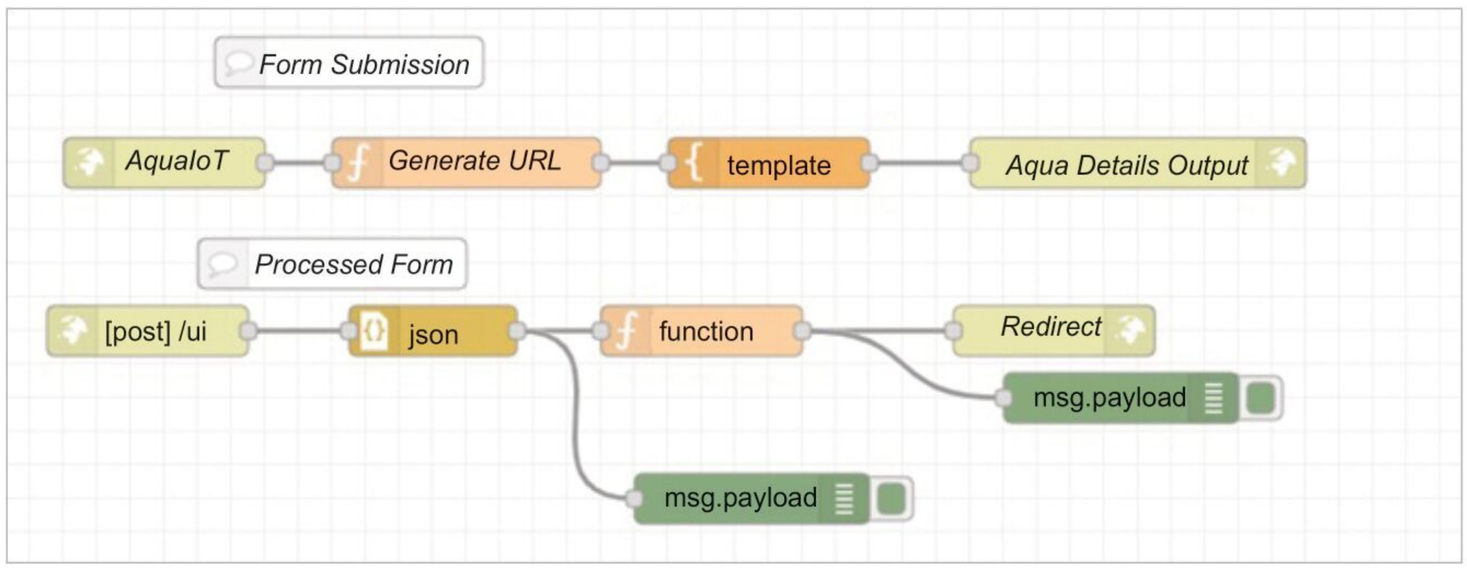
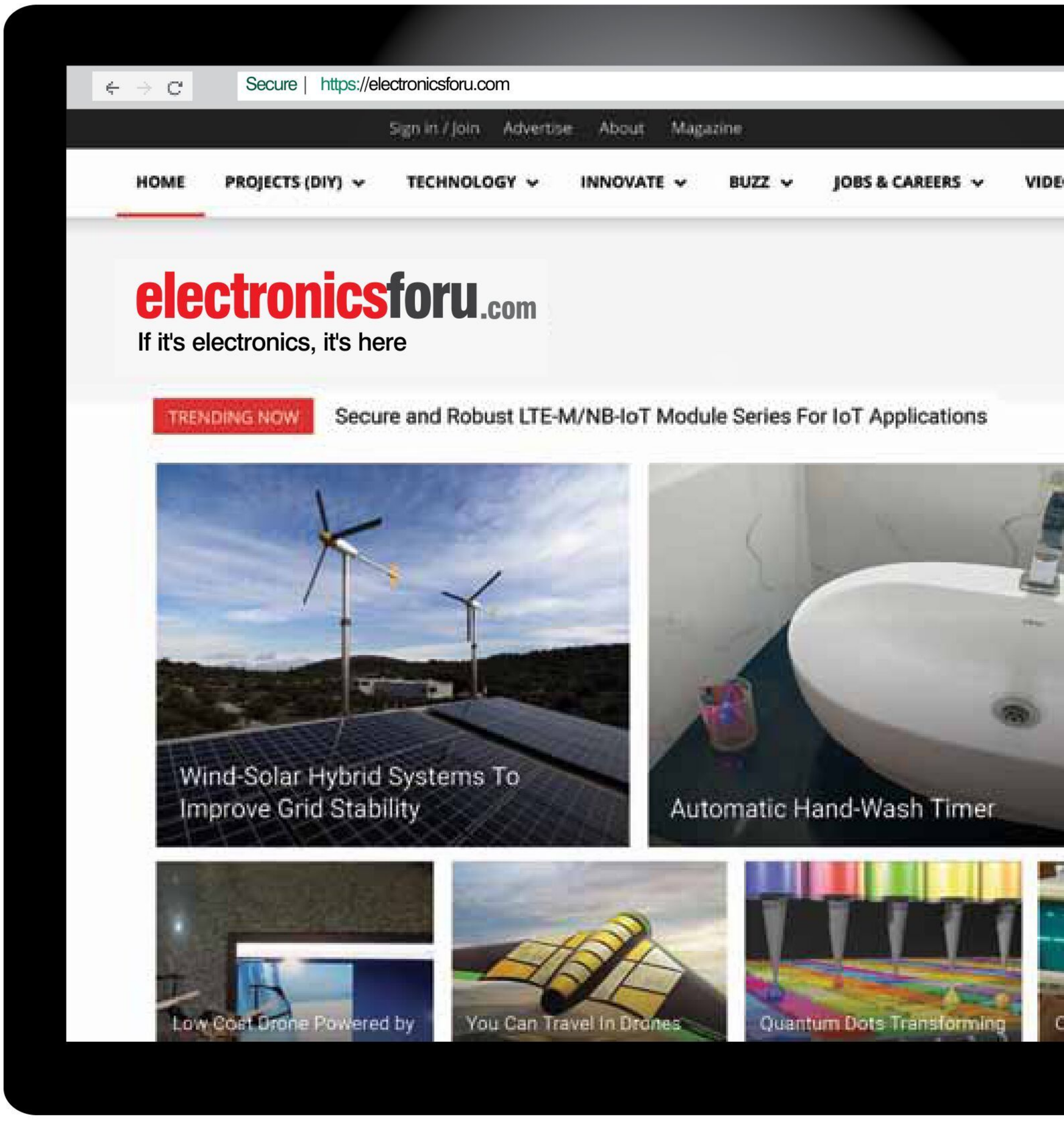


Figure 7: User interface design through Node-RED

Cloud integration: The data is sent to the cloud using the *cloudant out* node. Also, Node-RED is hosted in the cloud using IBM’s Cloudant database, as presented in Figure 6. The output dashboard is also hosted in the public network to allow remote access to the system using the mobile or Web.

Your favourite website has



electronicsforu.com

THANKS TO YOU—OUR ONLINE NETWORK IS

FACTS & FIGURES

- 4 websites (two more coming soon)
- Five major Facebook communities
- Seven major LinkedIN groups & pages
- Million-plus active users (monthly)
- Million-plus reach through Facebook
- Fifty-thousand-plus industry connections through LinkedIN

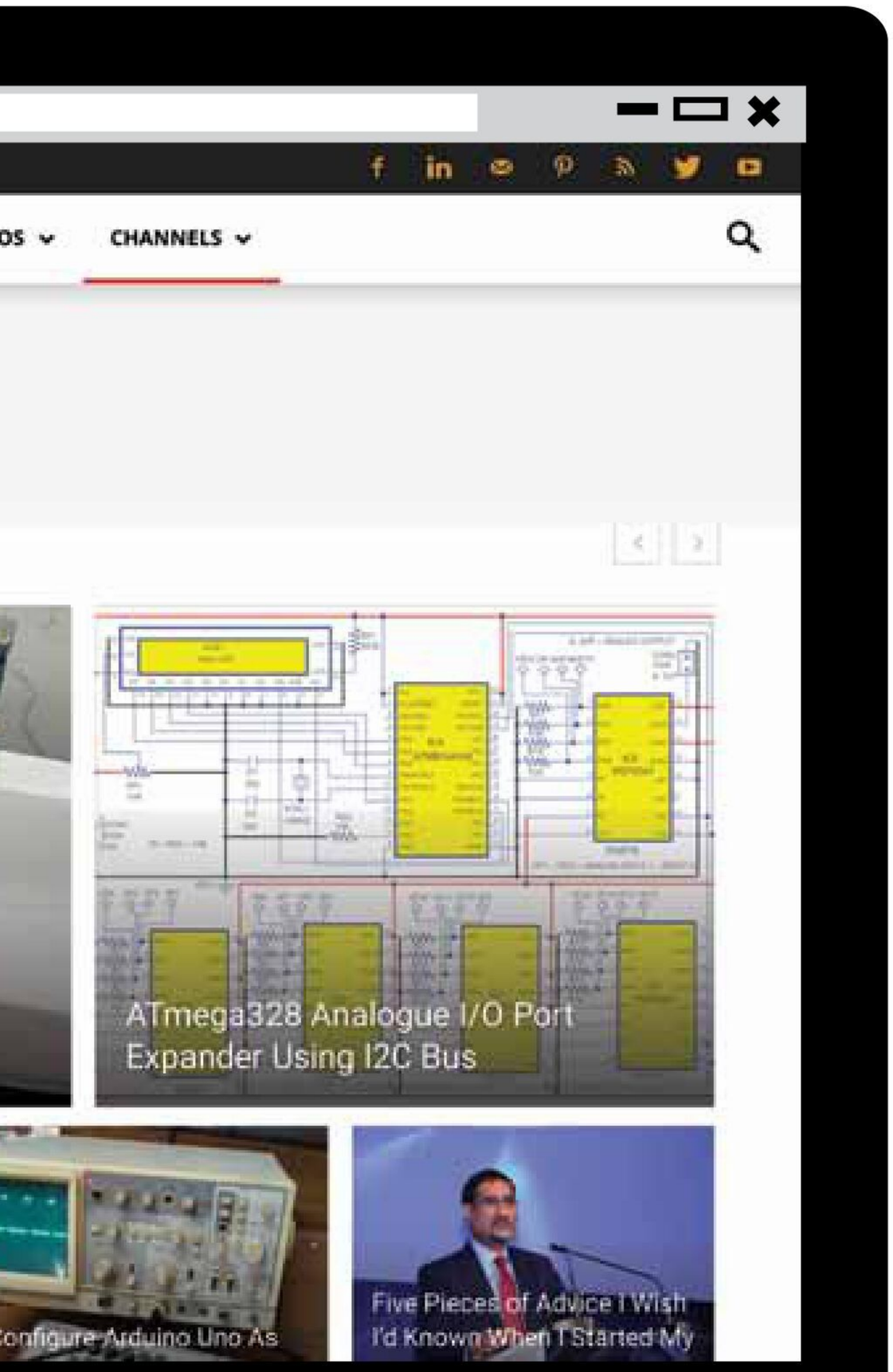
READERS

- You can access all content for FREE
- You can subscribe to newsletters for FREE --on most websites
- Register on our websites to get free invites to technical webinars and seminars

EXPERTS

- Experts who want to share their knowledge through articles, DIY Projects, etc are welcome
- We also welcome experts who want to share their knowledge through webinars or seminars
- You can contact us at editop@efy.in

fast growing peers now...



The Latest in IOT.
A platform for enablers, creators and providers of IOT solutions.

India. Electronics. Directory.
Enabling commerce between buyers & sellers of electronics in India.

Business. Electronics. India.
Everything you want to know about India's electronics industry.

Amazing DIY Projects. Latest Tech trends.

The hang-out for electronics enthusiasts.

AMONGST THE WORLD'S TOP 5 AND GROWING!

INDUSTRY

- You can advertise for as little as US$ 100 per month
- Special combo offers for advertisers in our print publications
- We've now enabled flexible CPM-based advertising
- You can advertise on the platform of your choice (based on your target audience)
- We invite press releases at efy-edit-team@efy.in
- Press releases are published free of cost, subject to discretion of the editorial team

RESPONSE GUARANTEED SOLUTIONS

We now also act as marketing partners for our clients and drive entire marketing for them, where we charge them on basis of results and not efforts!

CONTACT US: Shrikant Rao • growmybiz@efy.in • +91-98111 55335

Figure 8: User login page of iAquarium

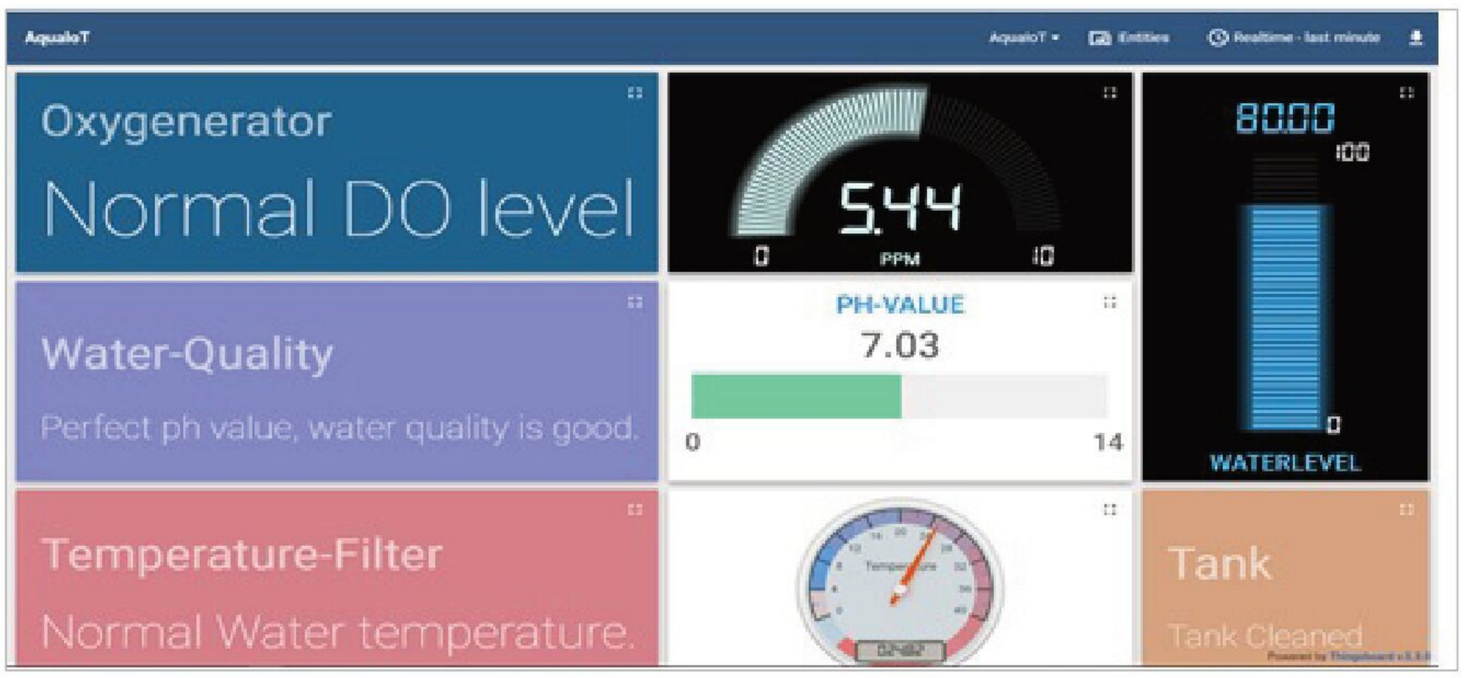


Figure 9: Dashboard visualisations of iAquarium

Web interface: This involves the creation of a Web interface, which consists of a login form to enter user details such as name, email ID, mobile number, and password. Once logged in, the user will be directed to view the status of the aquarium's tank. The UI design is simulated in Figure 7, and its outputs and dashboards are presented in Figures 8 and 9, respectively.

The idea here was to make a framework that can keep the fish in an aquarium healthy without carrying out manual monitoring of the water. This IoT based system enables consistent pH, water and temperature levels in an aquarium. This concept can be scaled for aquafarming. END

Dr Maheswari R.

The author is an associate professor at School of Computer Science and Engineering, VIT, Chennai.

Continued from page no...31

which shows logs coming out of the container, and a terminal window as well to manually run a few commands if and when needed.

All the communication from CLI to the cloud happens over a secured API, which you can also use to develop custom applications for interacting with your fleets and devices.

Where to go from here

The example above is a very basic hello-world example, but there is a lot more that one can do. This includes interfacing sensors, displays, actuators; defining a multi-container app with networking privileges, and exposing ports.

balena has amazing docs which can be found at *https://www.balena.io/docs*. You can also check out balena masterclasses for different balena tools at *https://www.balena.io/docs/learn/more/masterclasses/overview/*.

Two great projects made with balena, which you can try, are:

- balena Sense: *https://github.com/balenalabs/balena-sense*
- balena Dash: *https://github.com/balenalabs/balena-dash*

Some other great projects can be found at balenaHub, where community members have submitted their fleets, which you can join or fork. You can also learn about balena blocks, which are modular containers to ease functionalities like audio, Bluetooth connections, Chromium Web browser running from a container, and much more. You can check out all of this at *hub.balena.io*.

You can also join the balena community on the following channels:

- balena forums: *https://forums.balena.io*
- balena blog: *https://balena.io.blog*
- balena labs GitHub: h*ttps://github.com/balenalabs*

Hopefully, this tutorial will serve as a good starting point for anyone who wants to manage their IoT fleet efficiently. In case you have some questions, you can create a post on the balena forum, which is really active. END

Some other great projects can be found at balenaHub (*https://hub.balena.io)*

By: Ayan Pahwa

The author is an IoT developer advocate.

Choose the Most Optimal IoT Protocol for Your Project

The Internet of Things (IoT) is a term used to describe the interconnectivity and intercommunication of everyday devices with the Internet. Everything, from a smartwatch to a smart light bulb, is part of IoT. In this article, we talk about the different protocols used to enable easy communication between IoT devices. We also discuss which IoT protocol might be ideal for your project. If you are an IoT enthusiast looking to get started with your own project, this article is a must read.

It all started in 1982 with a Coca Cola vending machine developed at Carnegie Mellon University. Designed on the back of ARPANet, this machine had the ability to report the number of Coca Cola cans still left in the machine and whether these were cold. This basic idea of combining the Internet with regular day-to-day devices or 'things' took root and resulted in the birth of what is today known as the Internet of Things (IoT). The term IoT itself was coined in 1999 by Kevin Ashton, the co-founder of the Auto-ID center at MIT.

Since then, IoT has come a long way, evolving into a huge industry with millions of active devices serving a broad spectrum of applications. Smart home devices such as Google Home and Amazon Alexa are good examples of how IoT is being deployed by major tech companies to provide a wide range of user-friendly services. From smart TVs and refrigerators to smart surveillance cameras and lights, IoT is everywhere today.

As shown in Figure 1, by 2025, over 75 billion active IoT devices are estimated to function across the globe. With the AI revolution well on its way, IoT plays a key role in the future of electronics. Everything, from self-driving cars to automated train and aeroplane systems, has been made possible with IoT. Major tech companies such as Intel, Dexcom and Cisco have invested heavily into developing IoT devices and circuits. If you are looking to get started with getting your feet wet in the field of IoT, there was never a better time than now.

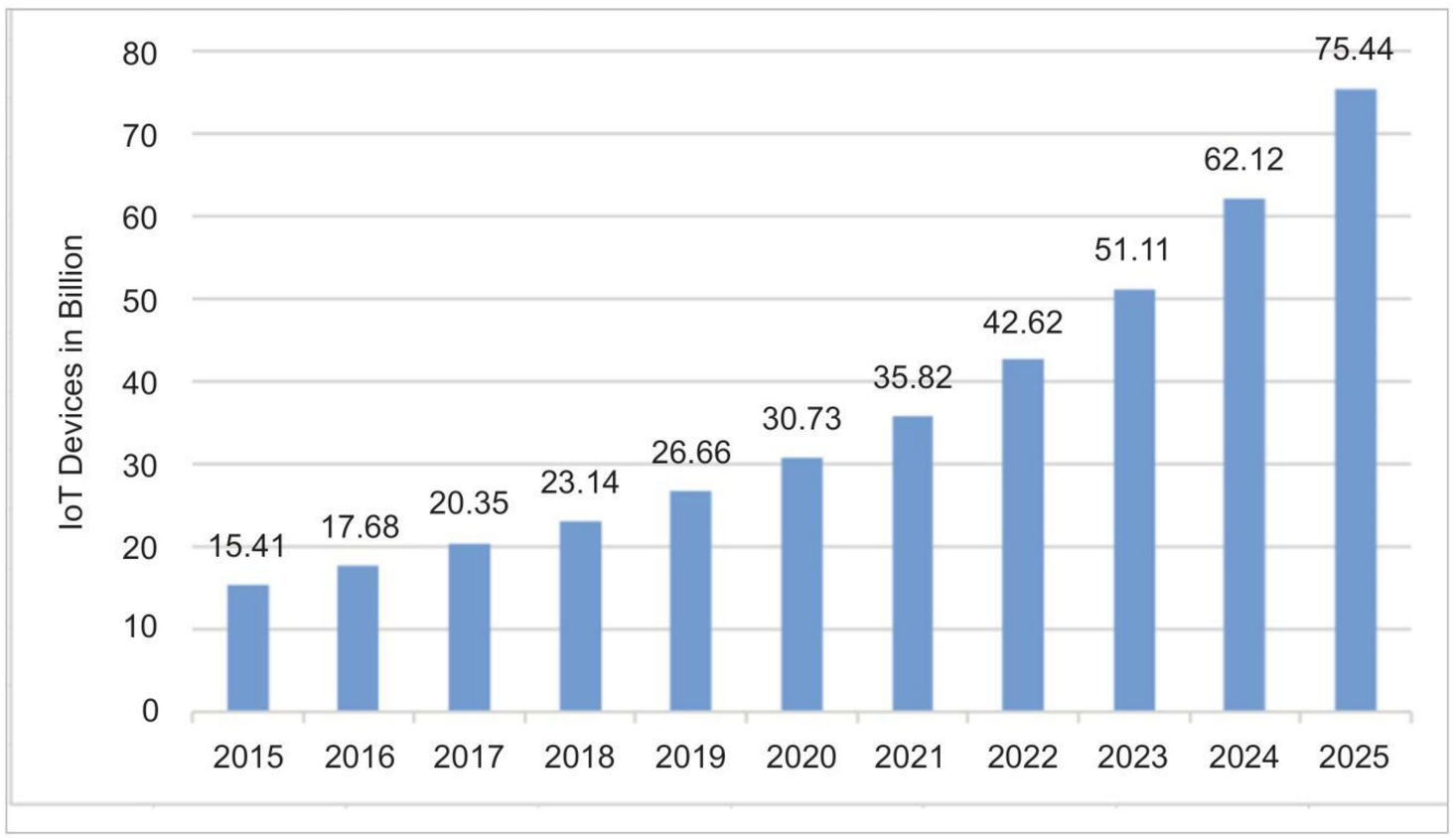


Figure 1: Projections of growth in IoT devices by 2025 (Courtesy: ResearchGate)

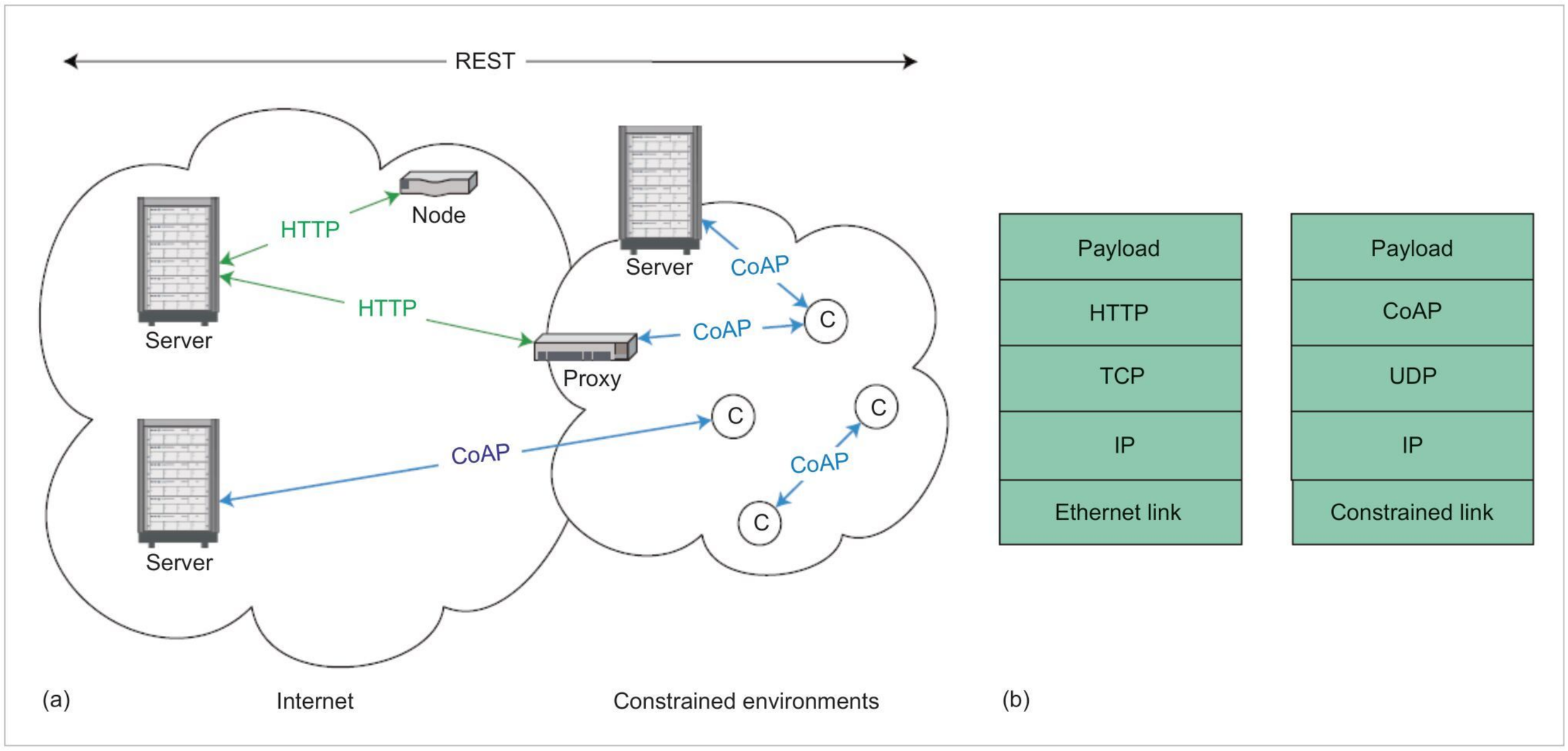


Figure 2: CoAP data transaction model (Courtesy: Devopedia)

Why do we need different protocols for IoT devices?

The rapid development of the Internet and the associated technologies that go with it has led to a surge in IoT devices. Almost everything, from your spectacles to your shoes, is now connected to the Internet. But not every device has the same processing capacity. The technology used in any wearable Apple device is far superior than, say, a basic pedometer. Add to that, Internet connectivity still remains a major issue across the globe. Even powered with the latest 5G Internet, your device is likely to suffer some sort of connectivity issues.

There are also some product-specific issues that developers come across such as the size of the device, which limits the number of chips you can fit into it. Take for instance, the latest wireless Bluetooth earbuds. The Bluetooth connectivity chips used in those devices need to be small enough to fit inside an ear and strong enough to stay connected to your phone or computer at a longer distance.

While such limitations have been a good thing, in the sense that they fuelled faster development of revolutionary technologies, they have also given rise to new ways of communication over the Internet. Since all IoT devices cannot communicate with each other on the same capacity and capability, and have different system based requirements, we require different IoT protocols to enable smooth communication and transactions.

IoT protocols

Now that we understand why we require different IoT protocols, let us take a deeper look at each of the major ones in use today. The most prominent IoT protocols are CoAP, MQTT, XMPP, DDS, AMQP and LwM2M.

Constrained application protocol (CoAP)

When it comes to communication over the Internet, there is no protocol like the HTTP. With its underlying REST framework and simplicity, HTTP is highly reliable and easy to implement. The trouble however is that it can prove heavy for most IoT devices, as these generally require a more lightweight protocol. So can there be a protocol that has all the goodness of HTTP in addition to being lightweight and easy to implement? That is the problem that the IETF Constrained RESTful environments working group tried to solve by creating CoAP in 2013. CoAP uses the user datagram protocol (UDP) for establishing communication between IoT devices, unlike HTTP which uses TCP.

CoAP uses a RESTful architecture and uses the HTTP GET, POST, PUT and DELETE methods. The beauty of CoAP, which distinguishes it from HTTP, is that UDP allows the IoT devices to communicate without having to establish a prior connection. This means that an IoT device using CoAP can relay multiple messages to multiple devices without having to wait for a handshake. This makes the communication faster while continuing to use very low bandwidth. To mitigate the risk of messages getting dropped due to a lack of connection, CoAP uses a confirmable/non-confirmable feature, using which messages that have been delivered can be marked confirmable. CoAP also uses datagram transport layer security (DTLS) for secure exchange of messages in the transport layer. Figure

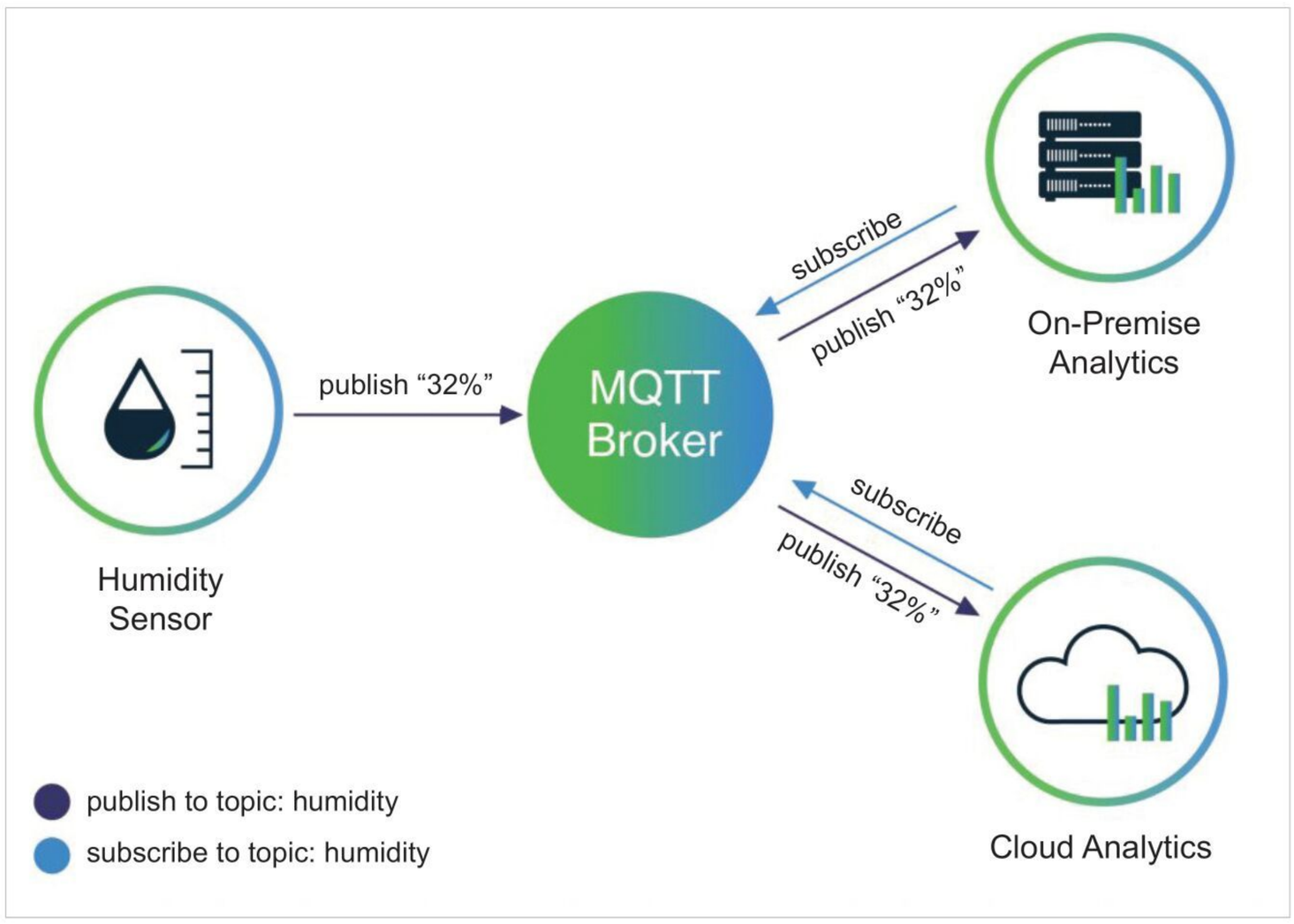


Figure 3: MQTT data transaction model (Courtesy: BehrTech)

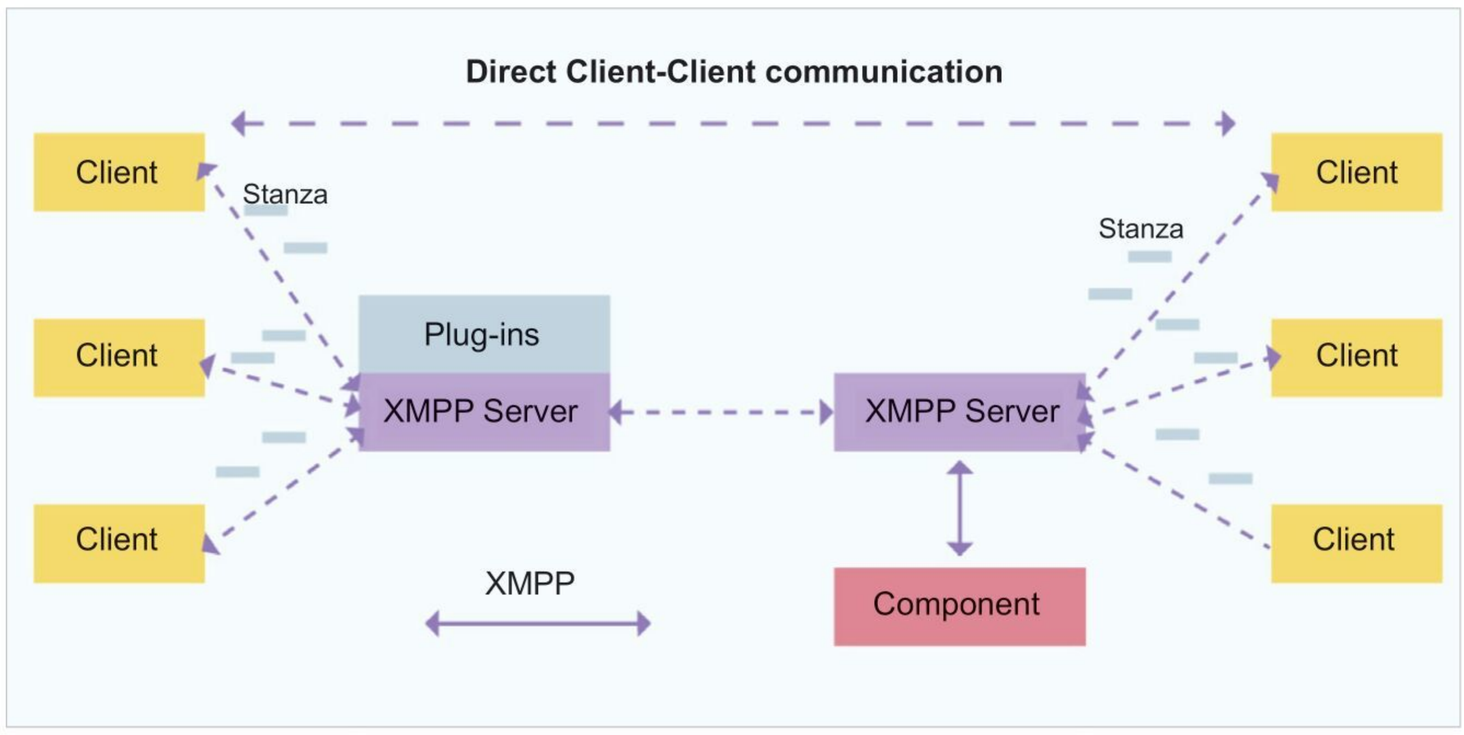


Figure 4: XMPP data transaction model (Courtesy: CometChat)

2 depicts the data transaction model of the CoAP protocol.

Message queuing telemetry transport (MQTT)

Perhaps the most widely used protocol in industry level IoT products, MQTT uses a publish/subscribe model for transacting data between different devices. In this model, devices are divided into three types, namely, the producers, consumers and brokers. Producers publish data under what are called topics. A topic is basically a channel under which related data can be published by the producer. The consumer devices subscribed to that particular channel then receive the message. The entire transmission of data from a publisher to its subscribers is made possible with the help of brokers. Brokers receive messages from producers that they then filter into designated topics and share with the subscribers for each of the topics.

MQTT uses the TCP/IP protocol to enable communication between multiple devices. What makes it ideal for industry level products is that it simplifies sending and receiving messages between multiple devices. Figure 3 depicts the data transaction model of MQTT with the help of an example of a smart humidity sensor. In the example, the humidifier is the producer that publishes data to the broker under separate topics. The consumers, which in this example are the on-premise analytics and the cloud, receive the data from the broker for their respective topics. So the producer publishes the data to the broker, and the broker publishes the data to the subscribers.

Extensible messaging and presence protocol (XMPP)

XMPP started as an instant messaging communication protocol but soon evolved into becoming an IoT protocol. It is an XML based server/client model protocol. The architecture, as depicted in Figure 4, is fairly straightforward. Clients connect to an XMPP server to communicate. The server facilitates the communication sessions between the clients. Since the medium of communication is in XML, it reduces the size of the data being transmitted and makes transactions faster.

There are, however, a few drawbacks in XMPP that do not make it an ideal solution for enterprise level IoT applications. For one, there is no quality-of-service feature in XMPP unlike CoAP, which makes it unreliable. Second, it doesn't have end-to-end encryption, which makes it insecure for sensitive data transmission.

Data distribution service (DDS)

DDS is similar to MQTT, but better. It follows a publish/subscribe model but does not require a broker for the transaction of data between devices. Add to that, it also supports multicasting and quality-of-service, which makes it ideal for usage in industry level products. The Object Management Group (OMG) first designed DDS with scalability in mind. Hence the brokerless model makes it highly scalable and adaptable to changing requirements.

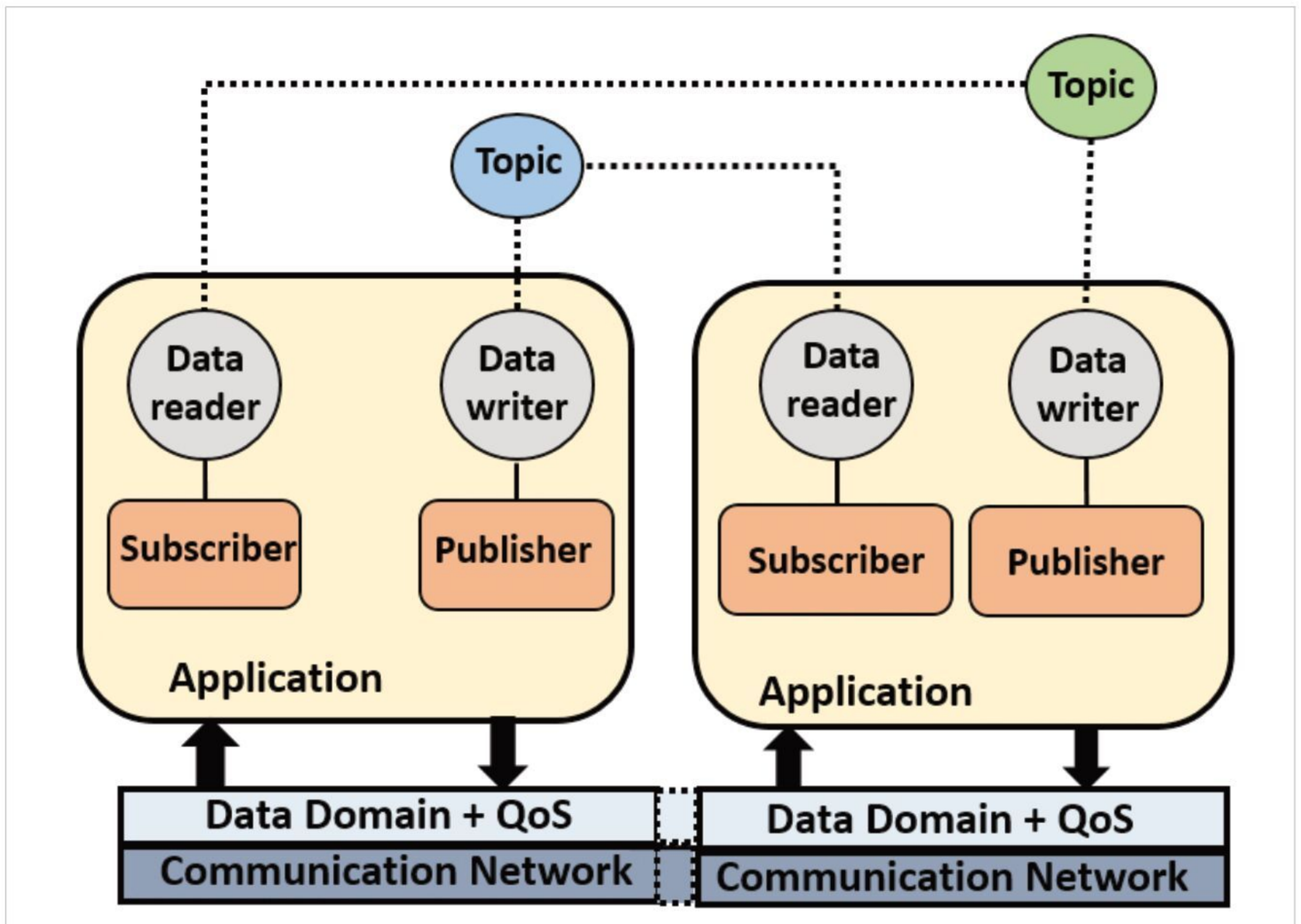


Figure 5: DDS data transaction model (Courtesy: ResearchGate)

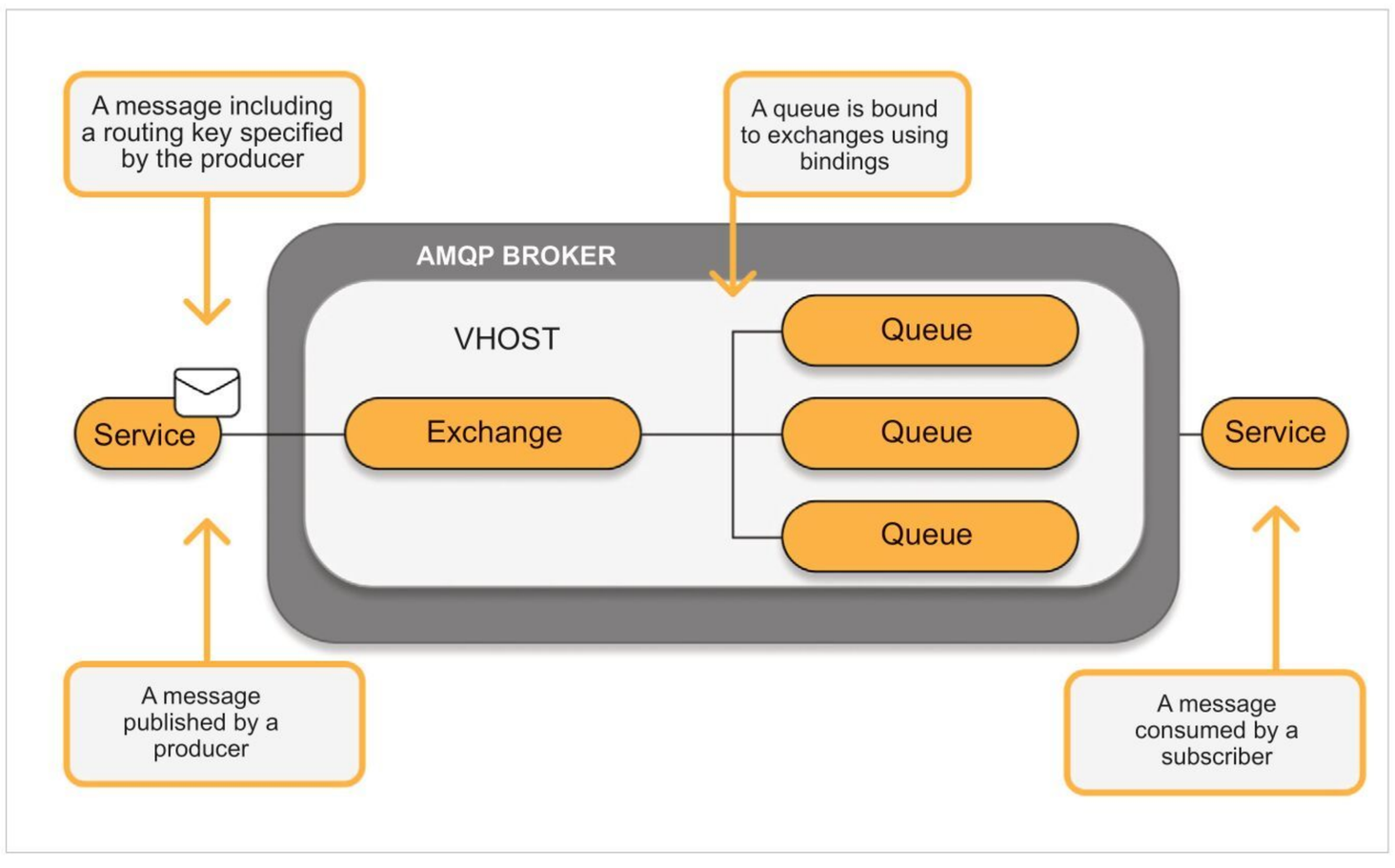


Figure 6: AMQP data transaction model (Courtesy: CloudAMQP)

As shown in Figure 5, the DDS architecture consists of a data-centric publish/subscribe layer (DCPL) in addition to an optional data-local reconstruction layer (DLRL). Scalability, resource awareness and fast paced distribution of data are ensured with the help of the DCPL layer, while the DLRL layer provides an interface for the operations of the DCPL layer.

Advanced message queuing protocol (AMQP)

AMQP is another publish/subscribe model based protocol that is primarily focused on guaranteed completion of transactions. Designed back in 2003, it was used mostly in the financial sector for obvious reasons. AMQP makes use of advanced queuing and distribution techniques to ensure security, reliability and speed.

As seen in Figure 6, the publisher publishes the message to the AMQP broker with a predesignated routing key attached to the message. Utilising the routing key, the broker queues the message for delivery to the designated consumer. These messages are bound to exchanges with the help of bindings. There are, however, a few drawbacks of the AMQP model which has restricted its mass adoption in the IoT space. For one, it is a heavy protocol that is not ideal for smaller IoT devices with a lack of bandwidth and power. Second, the guaranteed delivery of messages is not always a high priority for all IoT projects.

Lightweight M2M (LwM2M)

Designed by the Open Mobile Alliance (OMA) in 2014, LwM2M has been specifically designed to tackle the challenges of modern day IoT systems. With over 75 billion working IoT devices estimated across the world by 2025, LwM2M has been designed for scalability, speed and reliability in a constrained environment.

As shown in Figure 7, the LwM2M server stack consists of all the essential components that an IoT network requires including CoAP protocol, UDP protocol, SMS bearer and an efficient payload system. It also has DTLS security inbuilt. Additionally, LwM2M provides a device level bootstrapping facility, which gives you the ability to customise the usage of the protocol as per your requirements on a local level. It also provides a client registration interface, and access to resources and instances. This gives the user access to customise the settings on a granular level without having to revert to factory settings.

Which IoT protocol is right for your project?

What is interesting about IoT, unlike other tech projects, is that the requirements vary a lot from

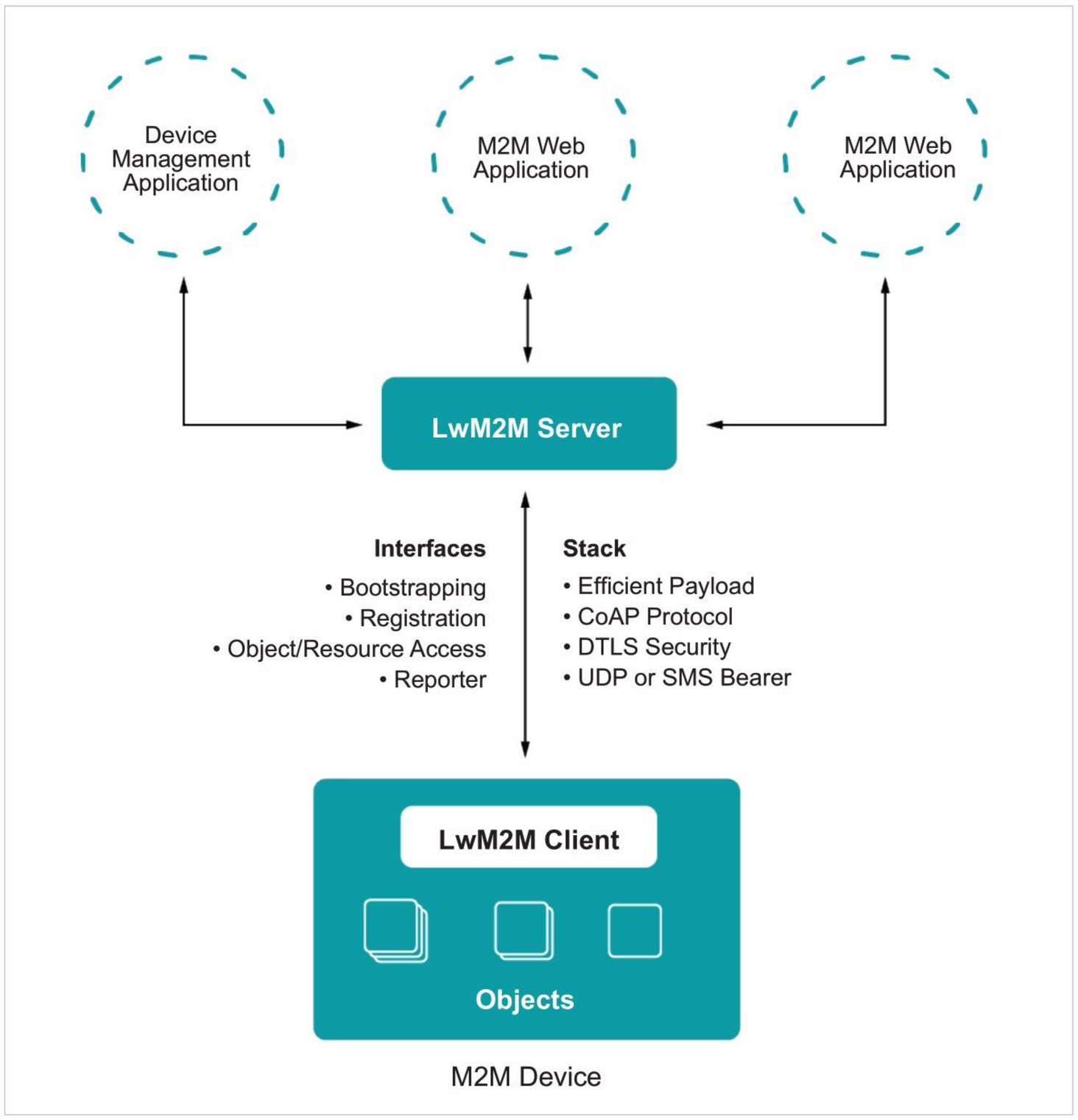


Figure 7: LwM2M data transaction model (Courtesy: AVSystem)

project to project, not only in terms of hardware but also software. For instance, an internal lighting system might require a connection with the cloud while another similar lighting system might require a more localised solution. In some projects, scalability and speed are the top priorities while for others, guaranteed completion of transactions and security might be the focus. As discussed in earlier sections, protocols play a critical role in facilitating such requirements. If the hardware is the body, the protocol is the brain of the device.

Keeping this in mind, I recommend doing a detailed audit and review of the project requirements. It is also important to research industry level IoT projects similar to yours and see which protocol they use. Here's a list of points to consider in this regard.

1. If using a RESTful framework with all the goodness of HTTP is a major requirement for your project, then you cannot go wrong with CoAP. CoAP provides all the features of HTTP, with the additional speed and reliability provided by UDP and message labelling.
2. If your IoT devices have major battery related constraints and you require a lightweight protocol, then MQTT is more suited for your project. It is important to keep in mind that MQTT does suffer with latency issues; so use this protocol only if high latency is not a major concern for your project.
3. If your project is data-intensive and requires fast transaction of messages between multiple devices, then XMPP might be a protocol worth considering. But do this only if the security of the data is not a major concern as XMPP does not have any inbuilt security features. Needless to say, if your project uses XML, then the go-to protocol should be XMPP.
4. If what you are working on are large, high-performance sensor networks such as the ones used in self-driving cars and smart air-traffic control systems, then DDS is probably the ideal choice. However, due diligence is necessary before selecting any protocol, and this is even more true when you are dealing with such advanced projects. DDS is a heavyweight protocol, and requires your devices to have a decent battery life and bandwidth to be able to function.
5. AMQP is a protocol that can be considered for projects that are end-to-end applications — for instance, heavy industrial machinery that has the capacity to handle data-intensive tasks without the concern for power consumption. If transaction completion is an absolute necessity for your project, then AMQP is the most reliable protocol for it.
6. LwM2M is the ideal protocol for telemetry and device management based IoT projects. You can create industry-level large scale IoT projects with high scalability, speed and security. LwM2M has been designed specifically for IoT projects and, as such, has a wide range of applications. You can consider using it for a wide array of projects with varying requirements and consumer bases. END

By: Mir H.S. Quadri

The author is a research analyst with specialisation in artificial intelligence and machine learning. A senior content developer at Arkinfotel, he loves analysing tech trends and understanding their implications. Being a FOSS enthusiast, he has contributed to several open source projects.

Using Node-RED for IoT and Edge Computing

All of us make use of IoT in one way or the other today, and this usage will only grow in the near future. This article details how Node-RED, a powerful and easy-to-use programming platform, helps in implementing IoT and edge computing.

The Internet of Things (IoT) is omnipresent in today's world. A few fields where it is dominant are listed below.

- **Smart health and medical services:** Smart ambulances, hospital management, intelligent drug control, etc.
- **Smart cities:** Smart traffic control, smart toll plazas, pollution monitoring, water quality management, self-driven cars, drones, law enforcement, energy conservation, etc.
- **Personal applications:** Smart health gadgets, theft avoidance, controlling home appliances, etc.
- **Retail sector:** Automated checkout, logistics monitoring and management, etc.
- **Agriculture:** Analysis of crops, dynamic water distribution, smart irrigation, farm surveillance, drones for smart farming, agriculture robots, etc.

Many other domains, too, are associated with the field of IoT, especially where intelligent robotic applications are being developed. The Internet of Everything (IoE) is another term used for smart

Figure 1: Node.js platform for a range of applications

applications, and is the integration of IoT with the cloud and the World Wide Web for the real-time connectivity of devices.

A number of programming platforms are available for working with IoT, IoE, fog or edge scenarios. Hundreds of toolkits, which are very powerful and easy to use for dynamic research, are available as well. A few of these are listed in Table 1.

Node-RED: A tool for flow based programming of IoT scenarios

Node-RED (*https://nodered.org/*) is a powerful and easy-to-use programming platform for the simulation of IoT scenarios. Fog and edge computing can also be done using flow based programming in Node-RED. Here, high performance structures can be implemented using minimum coding.

Installation and working

Node-RED is a specialised package that is installed onto the Node.js platform. The latter is a lightweight but high performance programming environment based on JavaScript. Many packages are available in Node.js for multiple applications, including Internet of Things (IoT), cloud computing, machine learning, data science, and blockchain.

To work with Node-RED, the Node.js platform should be installed first, which is available at *https://nodejs.org* for multiple operating systems like Windows, Mac and Linux for 32-bit or 64-bit architectures (Figure 1).

After the installation of Node.js, the Node-RED package can be installed from the node package manager (NPM), which is the repository of packages developed and deployed for the Node platform (Figure 2).

In the installation directory of Node.js, the *npm* command is used to install Node-RED, as follows:

```
E:\>cd nodejs
E:\nodejs>npm i node-red
```

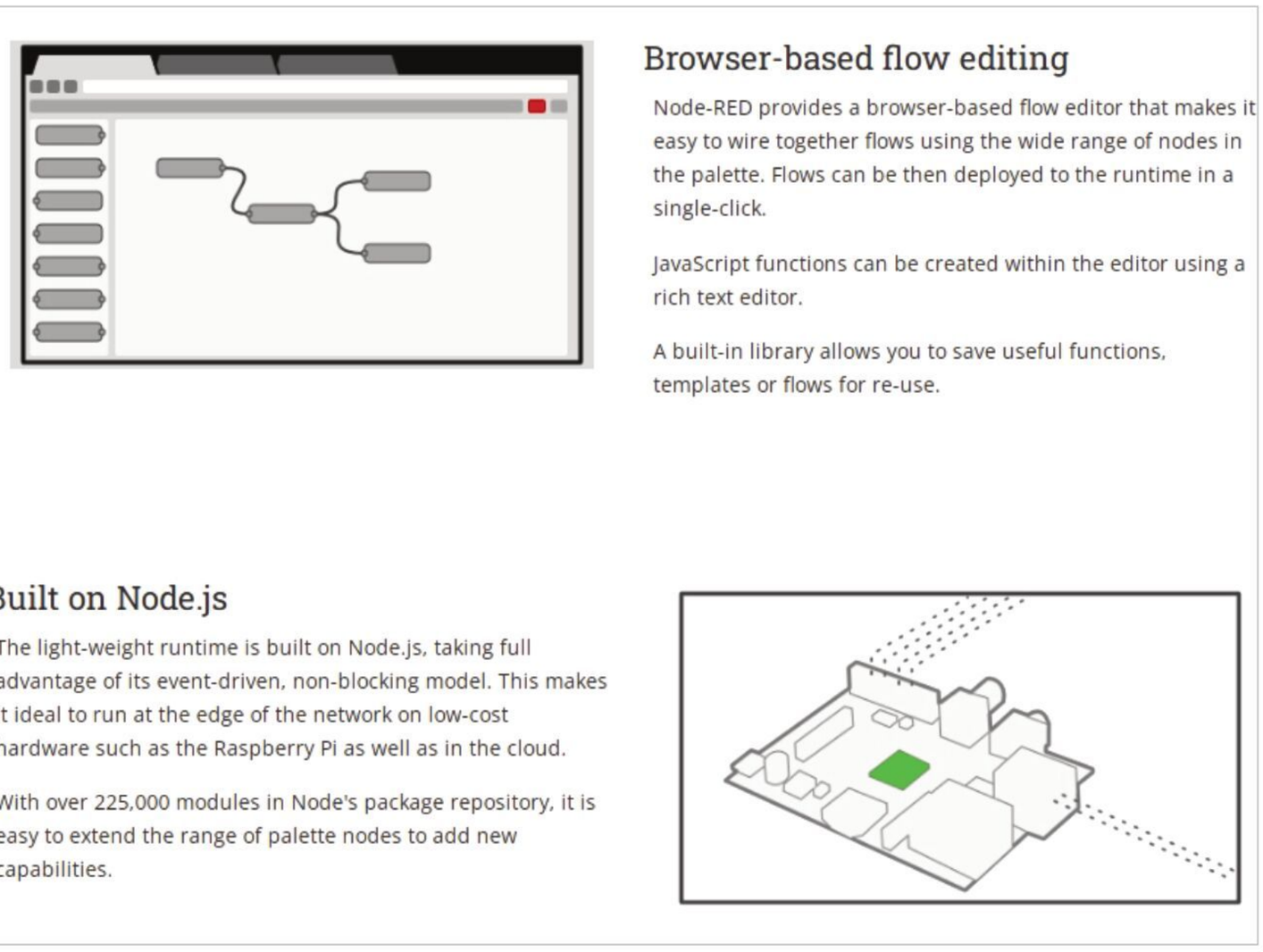


Figure 2: Node-RED platform built on Node.js (Source: *nodered.org*)

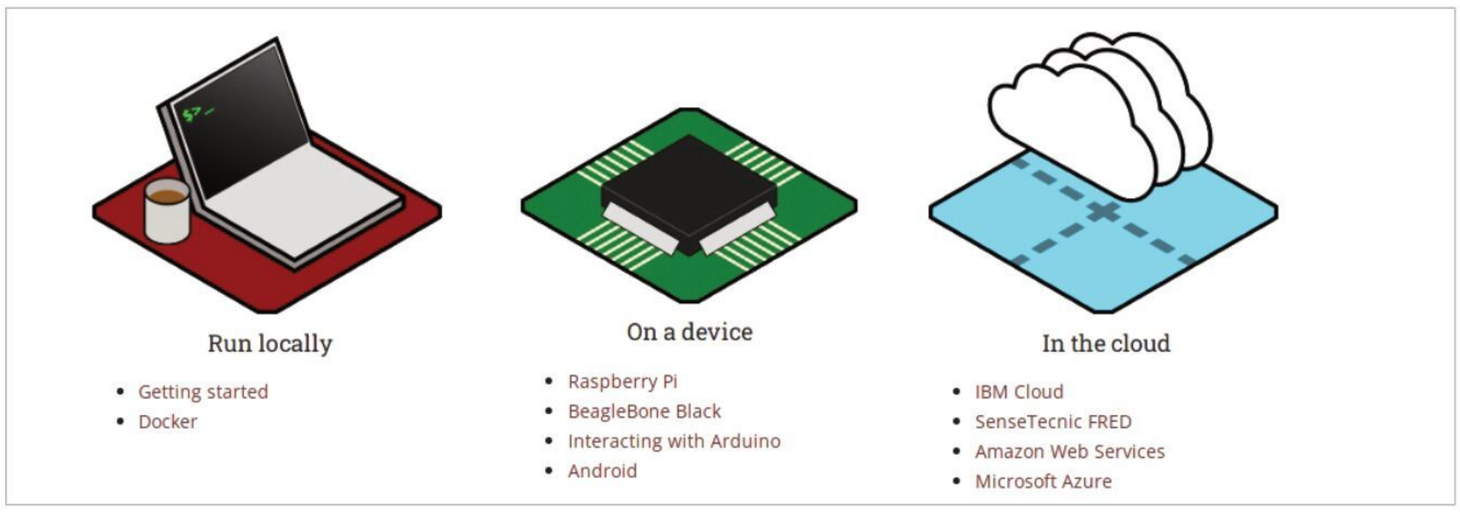


Figure 3: Installation options for Node-RED (Source: *nodered.org*)

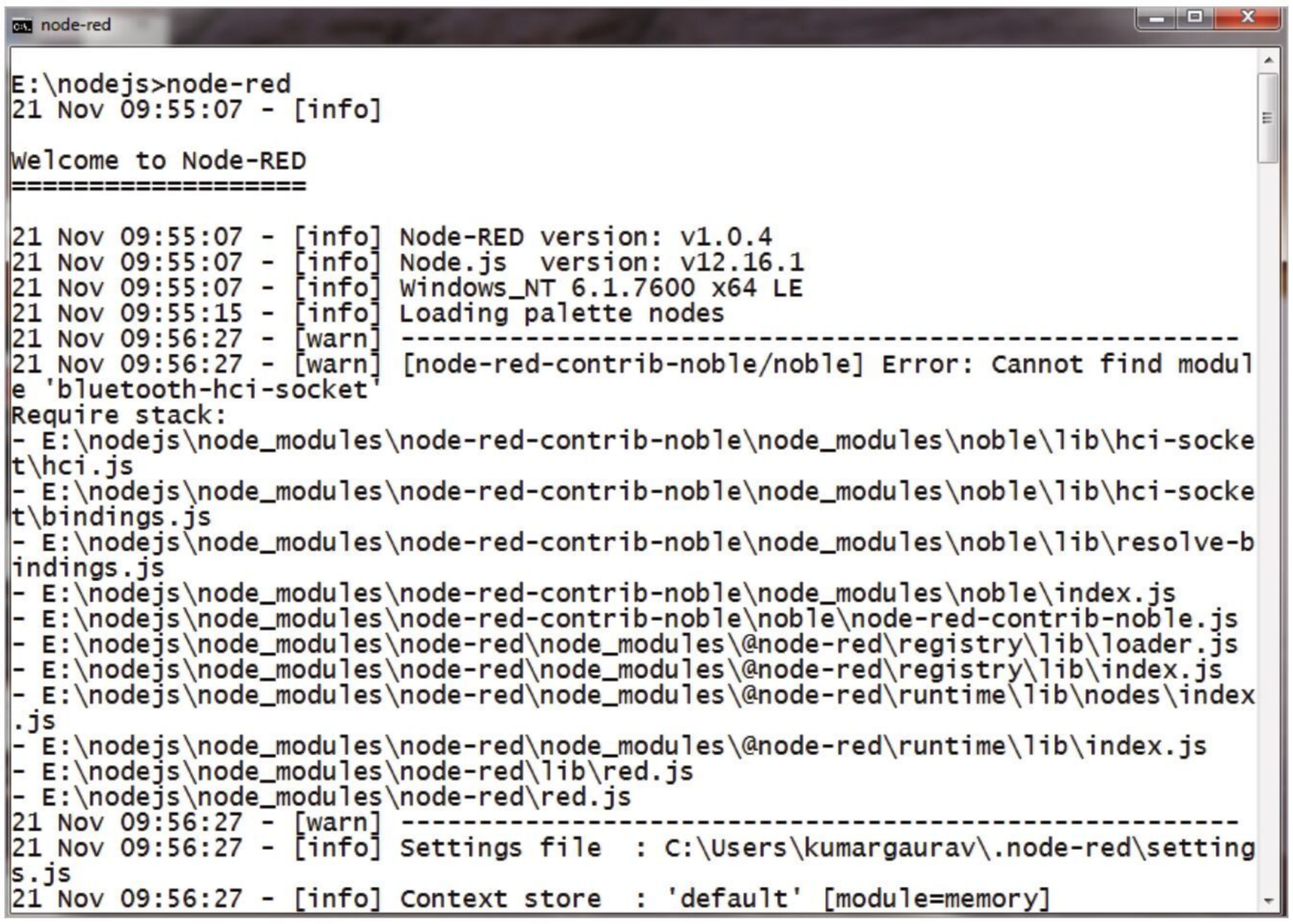


Figure 4: Starting the Node-RED server

Table 1: Platforms and toolkits for IoT and edge computing

Software	URL	Usage
Node-RED	*nodered.org*	Flow based programming environment
Contiki	*contiki-os.org*	Microcontrollers, IPv6, IPv4, protothreads, low resource, game consoles
FlowHub	*flowhub.io*	Flow based IoT programming
NoFloJS	*noflojs.org*	JavaScript based flow programming
Netron	*github.com/lutzroeder/netron*	Dynamic visualisation
PyFlow	*wonderworks-software.github.io/PyFlow*	Visual scripting
Yet another robot platform (YARP)	*yarp.it*	Robotic programming
OROCOS	*orocos.org*	Robotic programming and machine control
OpenIoT	*openiot.eu*	Sensing as a service (S2aaS)
Zetta	*zettajs.org*	WebSocket programming, real-time on TCP, reactive programming, low overhead scenarios
DSA	*iot-dsa.org*	Real-time interfacing, inter-device communication, programming on multiple layers
IoTivity	*iotivity.org*	Constrained application protocol (CoAP), IoT programming
CupCarbon	*cupcarbon.com*	Smart city, SCI-WSN simulation, visualisation, 2D and 3D OpenStreetMap, MQTT programming, sensor programming
KAA	*kaaproject.org*	Data analytics, real-time applications, dynamic communications and updates

Node-RED can be installed on multiple platforms including open source hardware devices, IoT devices, the cloud or on dedicated servers. Its cloud deployment can be done on IBM Cloud, Microsoft Azure, SenseTecnic FRED and Amazon Web Services. To work with IoT, fog and edge scenarios, it can be installed on Raspberry Pi, BeagleBone Black, or Arduino. These devices support the interfacing of IoT sensors for multiple applications.

To run Node-RED on a dedicated system, the *node-red* command is executed to start the server. A local host based IP address with port number is generated, on which the programming for IoT, edge computing, etc, is done.

After starting the server, there are thousands of packages that can be interfaced for multiple applications. To create an IoT or edge scenario, packages related to Wi-Fi, Bluetooth, MQTT, CoAP, weather forecasting, etc, can be installed from the NPM repository.

After the required packages are installed, the flow diagram is created

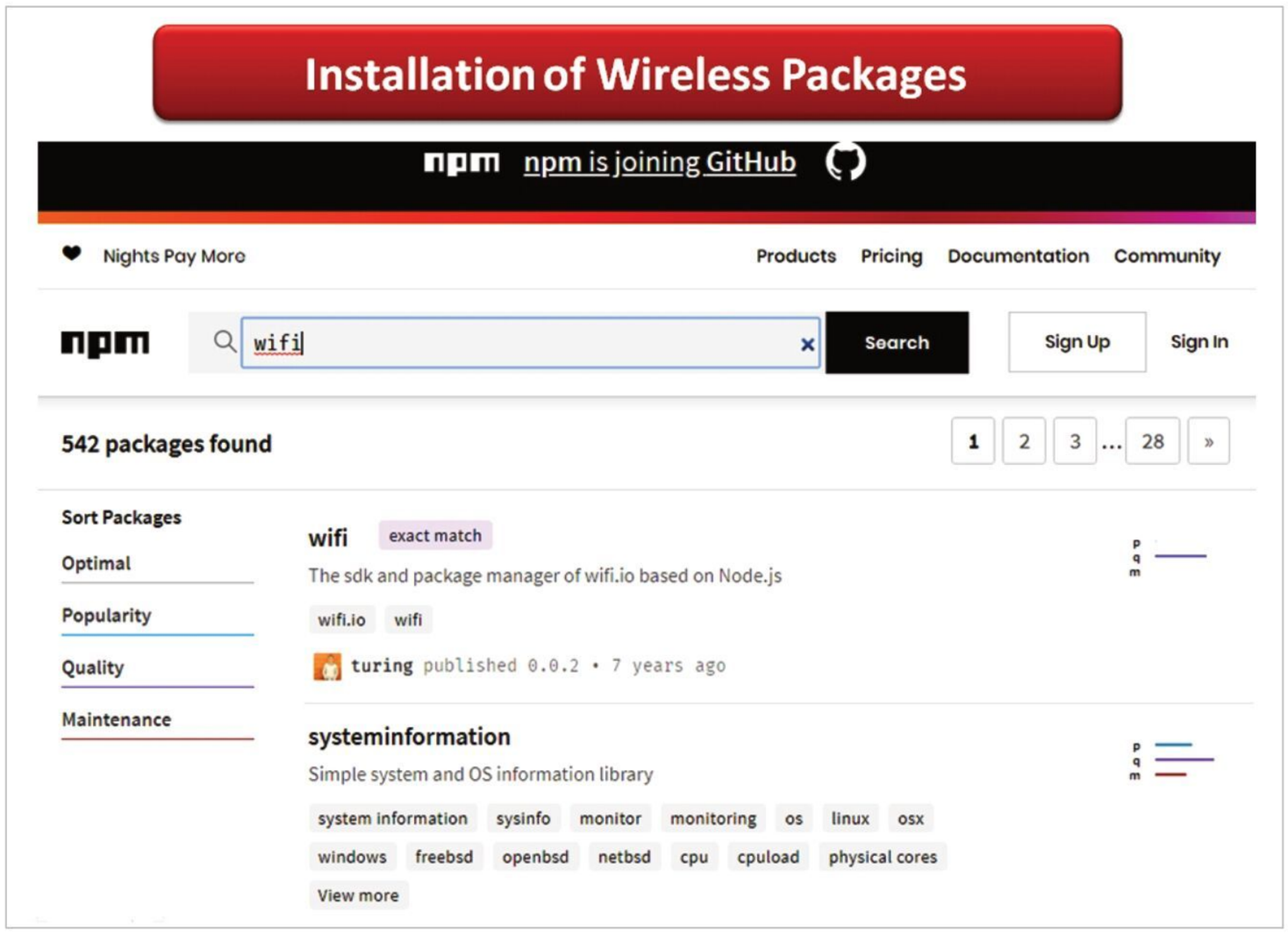


Figure 5: Searching a Wi-Fi package from the NPM repository

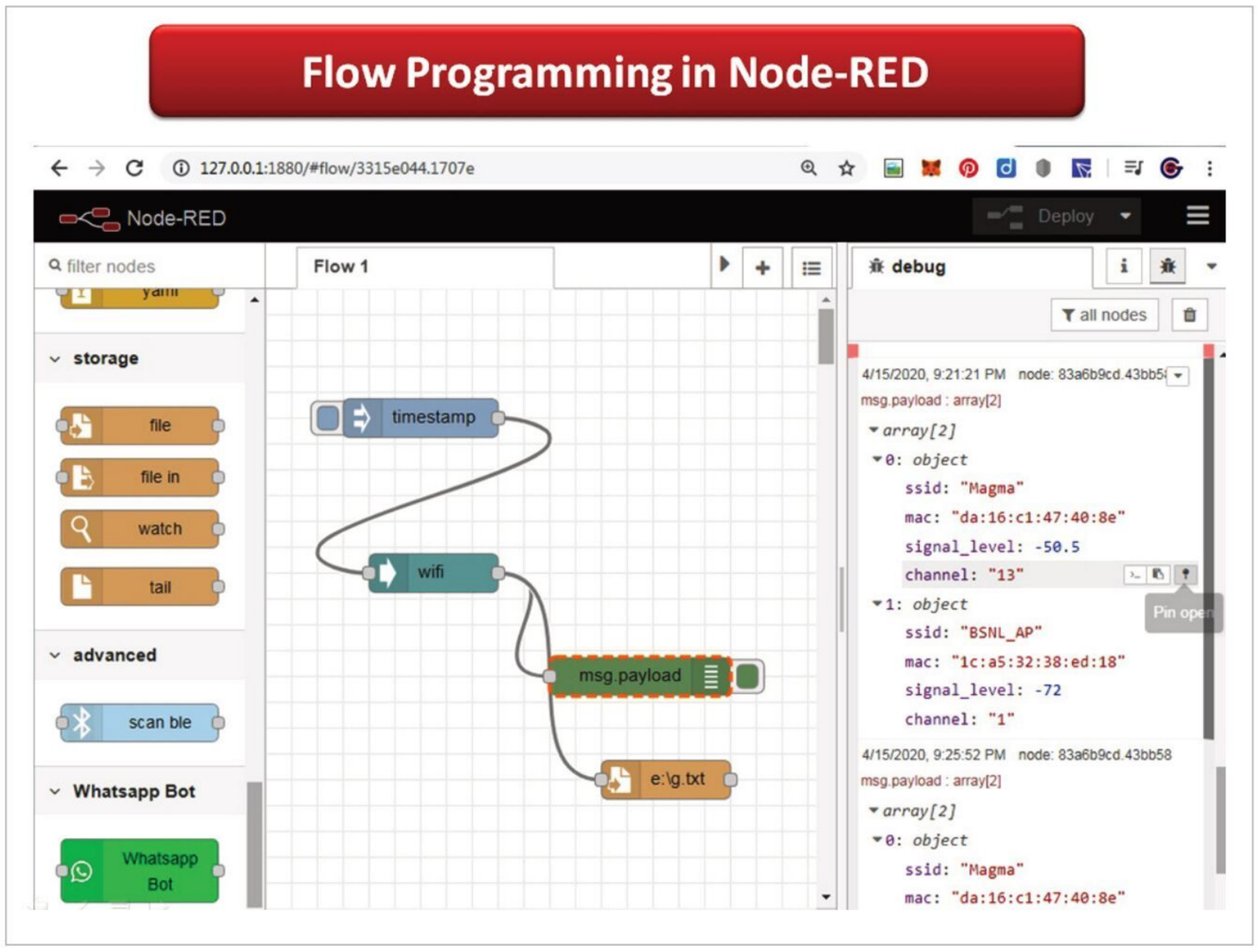


Figure 6: Creating a flow diagram in Node-RED with Wi-Fi interfacing

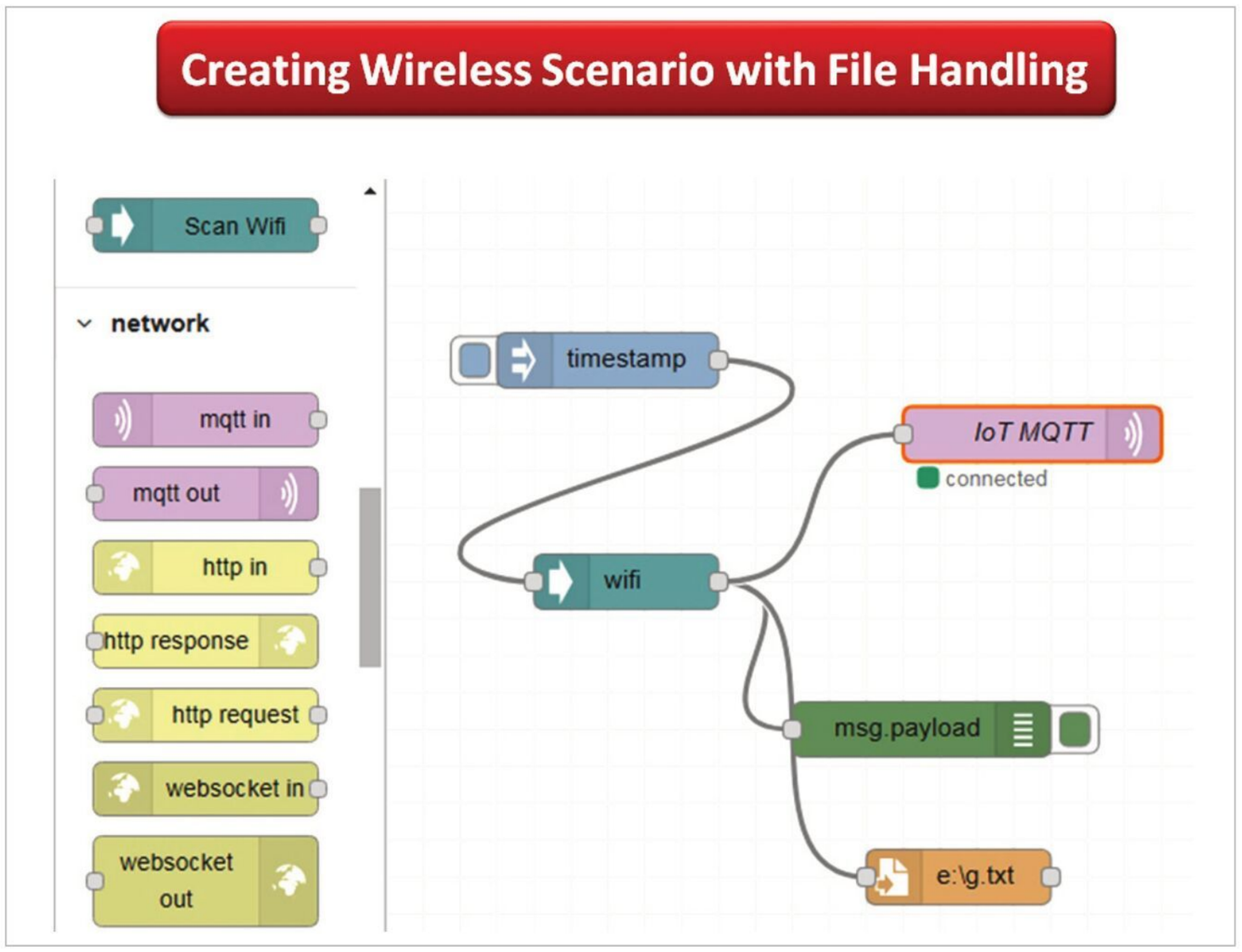


Figure 7: Creating an IoT scenario for real-time interfacing with MQTT broker

using the left palette of icons. Here, all the packages are visible and can be dragged across the dashboard.

The message queueing telemetry transport (MQTT) protocol is available in the Node-RED palette to help it work with IoT. It is used as an IoT broker for real-time interfacing and communication with sensors. An IoT scenario is created by the file handling module, capturing data from sensors and using the MQTT protocol to interface it with Wi-Fi. After running this scenario on a single click, the results are visible on the right side panel of Node-RED. Similar other structures can be created and deployed for dynamic communication with IoT devices.

Researchers and engineers can create IoT scenarios as per their requirements with the customisation and addition of packages on Node-RED.

Node-RED can interface with Raspberry Pi or Arduino to provide real-time communication with sensors, which can be used for smart agriculture based applications so that farmers can keep track of their fields and crops. Weather forecasting and climate analysis can also be done using such programming platforms. END

By: Dr Gaurav Kumar

The author is associated with various academic research institutes, where he delivers expert lectures and conducts technical workshops on the latest technologies and tools.

THE COMPLETE MAGAZINE ON OPEN SOURCE
OpenSource ForYou
The latest from the Open Source world is here.
OpenSourceForU.com
Join the community at facebook.com/opensourceforu
Follow us on Twitter @OpenSourceForU

An Introduction to IoT Protocols

Across the globe, billions of devices are today communicating and exchanging data with each other, thanks to the Internet of Things (IoT). IoT communication protocols protect and ensure security to the data being exchanged between these devices. This article introduces the most popular protocols in use today.

Things around us are not only getting smarter and intelligent, but also connecting and interacting seamlessly with each other. The Internet of Things (IoT) refers to a collection of devices that are connected to the Internet, embedded with electronics, software, sensors, actuators, and network connectivity enabling these objects to collect and exchange data. The data gathered from IoT can be used to predict, learn, and make real-time decisions. IoT plays a significant role in applications like healthcare, smart homes, transportation, automation, agriculture, vehicles, and disaster recovery.

The IoT system can function and transfer information only when devices are online and safely connected to a communications network. This is where IoT standards and protocols make an entry. IoT devices can be connected either using an IP or a non-IP network. IP network connections are relatively complex and require increased memory and power from IoT devices, although range is not a problem. On the other hand, non-IP networks demand relatively less power and memory but have a range limitation.

IoT protocol architecture

IoT architecture depends on its functionality and implementation in different sectors. The basic process flow, on which IoT is built, has two important architectures -- the 3-layer architecture and 5-layer architecture.

3-layer IoT architecture: The 3-layer architecture is the most basic one. It comprises three layers, namely, the perception, network and application layers.

- The perception layer is the physical layer, which includes all the smart sensor-based devices that collect the data from the environment.
- The network layer includes all the wireless and the wired communication technologies, and is responsible for providing connections between the devices and the applications of the IoT ecosystem. The data is then passed on to the application layer.
- The application layer is accountable for delivering application-specific services to the user. It defines

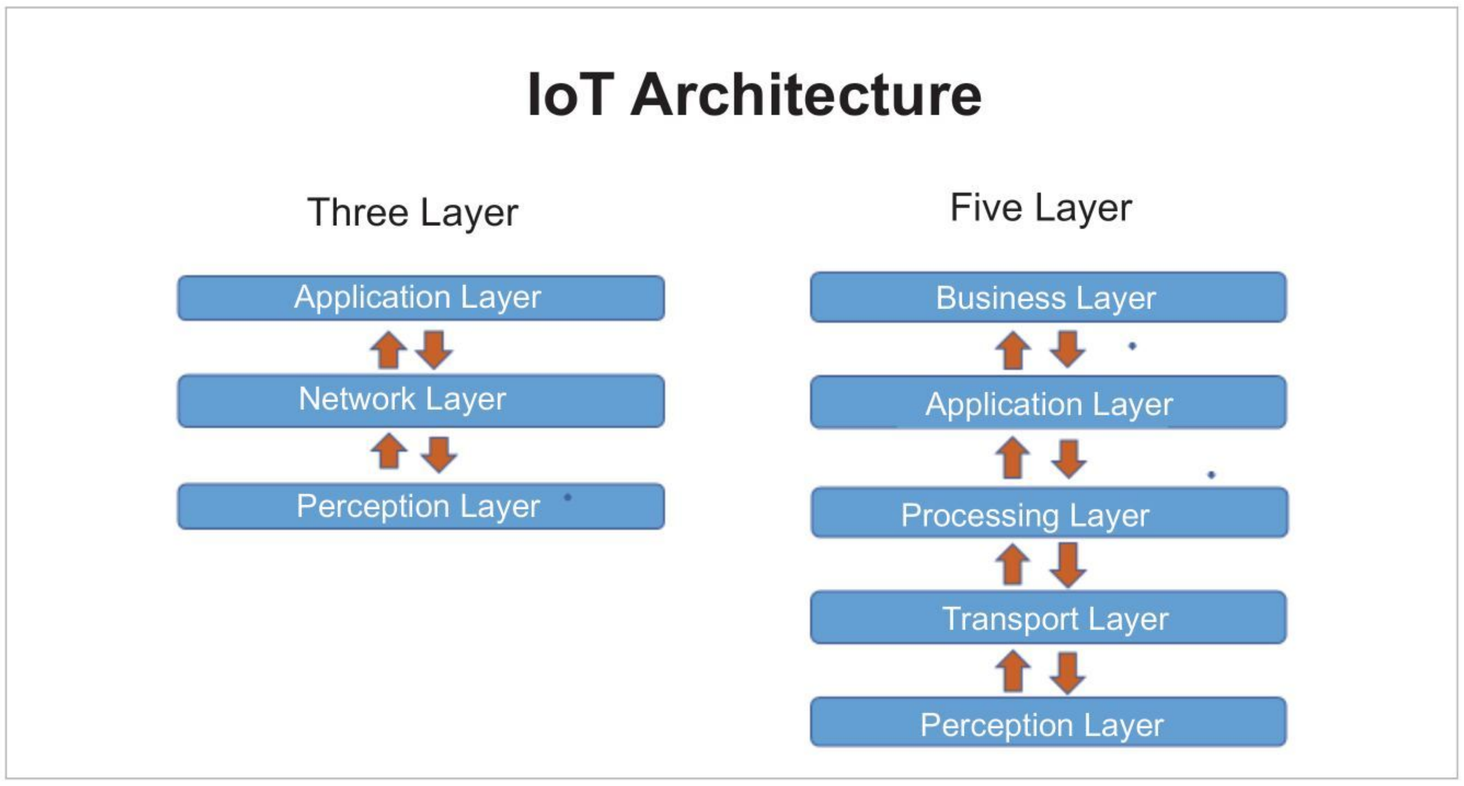


Figure 1: IoT architecture (Reference: *https://www.hindawi.com/journals/jece/2017/9324035/*)

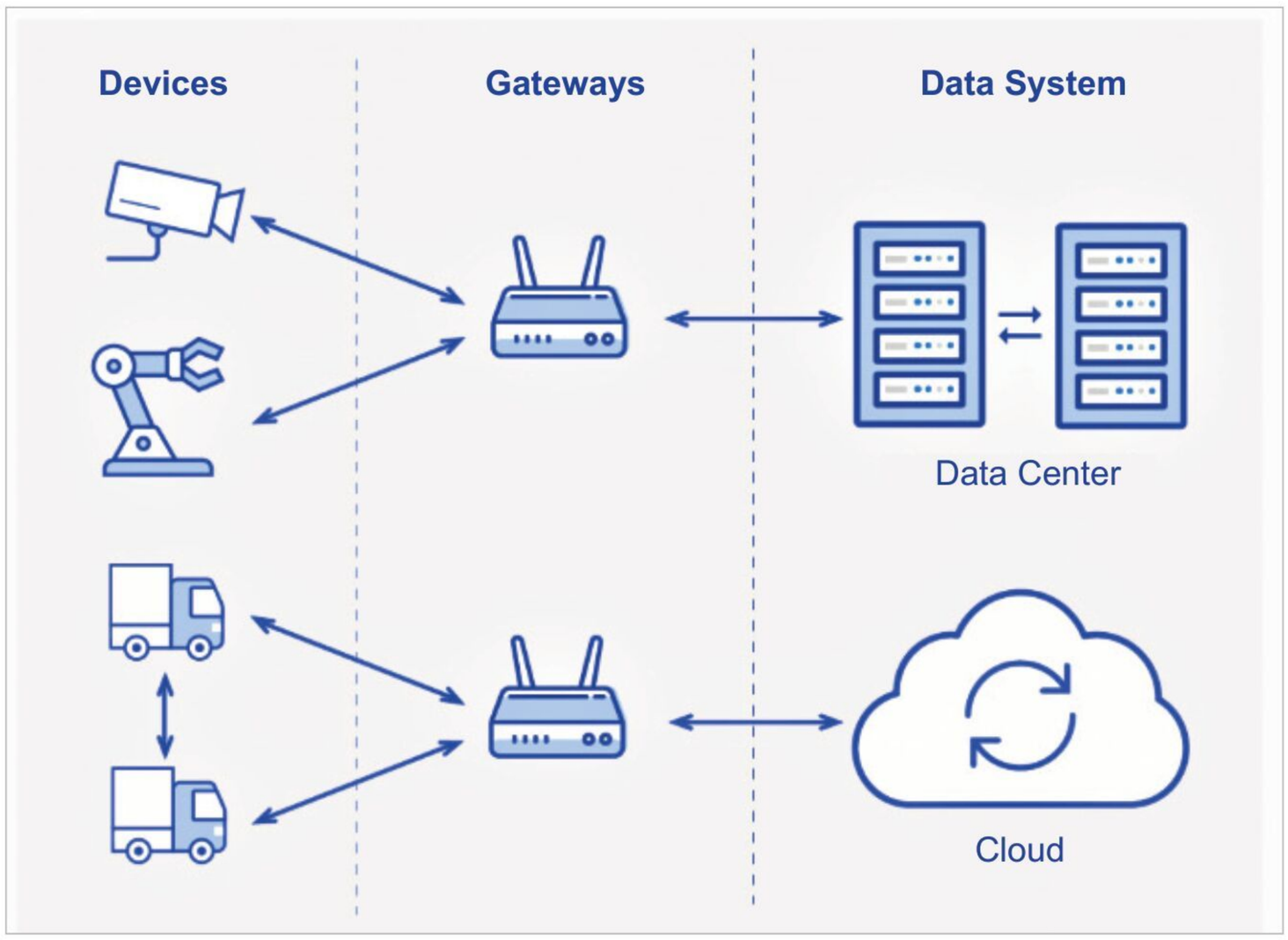


Figure 2: Types of IoT connections (Reference: *https://www.sam-solutions.com/blog/internet-of-things-iot-protocols-and-connectivity-options-an-overview/*)

various applications in which IoT can be deployed, like smart homes, cities, and health.

5-layer IoT architecture: The 5-layer architecture is an extension of the three-layer architecture with the addition of two more layers – the processing and business layers. The perception and application layers work in a similar manner as in the 3-layer architecture.

- The transport layer carries the sensor data from the perception to the processing layer and vice versa using networks such as wireless, Bluetooth, 3G, RFID, and NFC.
- The processing layer or middleware layer stores, analyses, and processes large amounts of data that come from the transport layer, utilising many technologies such as databases, cloud computing, and Big Data processing modules.
- The business layer manages the whole IoT system, including applications, businesses, and user privacy.

Types of IoT connections

An IoT system has four types of transmission channels for data communication.

Device-to-device (D2D) communication allows the devices located in close proximity to communicate with each other using Bluetooth, ZigBee, or Z-Wave protocols. To establish a connection without a network is possible through a D2D connection.

Device-to-gateway communication connects devices using an intermediary platform. Gateways serve two main functions – first, to consolidate data from sensors and route it to the relevant data system, and second, to analyse data and if any problems are found, return it back to the device.

Gateway-to-data systems communication is the data transmission from a gateway to the appropriate data system.

Communication between data systems is within data centres or clouds. For this type of connection, protocols should be easy to deploy and integrate with existing apps. They should have high availability, capacity, and reliable disaster recovery.

There are two types of IoT protocols.

Network layer protocols: IoT network protocols connect medium to high power devices over the network. End-to-end data communication within the network is allowed using this protocol. HTTP, LoRaWAN, Bluetooth, Zigbee are a few popular IoT network protocols.

IoT data protocols: IoT data protocols connect low power IoT devices. Without any Internet connection, these protocols can provide end-to-end communication with the hardware. Connectivity in IoT data protocols can be done via a wired or cellular network. MQTT, CoAP, AMQP, XMPP, DDS are some popular IoT data protocols.

IoT protocols and network standards

There are many IoT protocols available for different applications and requirements. However, each has its own advantages and disadvantages for different IoT scenarios. Some of the most widely used IoT protocols are discussed here.

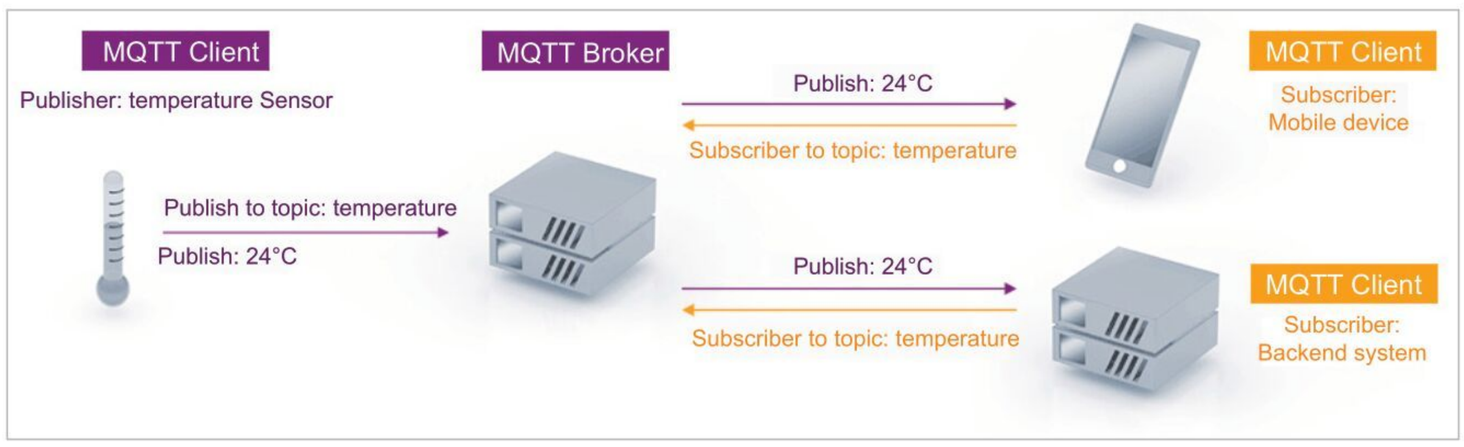


Figure 3: MQTT publish/subscribe architecture (Reference: *https://mqtt.org//*

Message queue telemetry transport (MQTT) protocol

This open source publish/subscribe messaging transport protocol is very lightweight and ideal for connecting small devices to constrained networks. It was designed to work in low-bandwidth conditions, such as sensors and mobile devices, on unreliable networks. This capability makes it the most preferred protocol for connecting devices with small code footprint, as well as for wireless networks with varying levels of latency due to bandwidth constraints or unreliable connections. It works on top of transmission control protocol/Internet protocol (TCP/IP) to provide reliable delivery of data. MQTT has three main components:

- Subscriber
- Publisher
- Broker

The basic workflow of this protocol involves the publisher being responsible for generating and transmitting information to subscribers through a broker. The main function of the broker is to ensure security by checking the authorisation of subscribers and publishers. This protocol is preferred for IoT devices because it provides well-organised information routing functions to small, cheap, low memory, power devices and low bandwidth networks. To ensure message reliability, MQTT uses three levels of quality of service (QoS).

- *QoS0 (At most once):* This is the least reliable but the quickest mode. The message is sent without any confirmation received.
- *QoS1 (At least once):* Ensures that the message is delivered at least once, even if duplicate messages may be received.
- *QoS2 (Exactly once):* This is the most reliable but also the most bandwidth-consuming mode. Duplicate messages are controlled to ensure that it gets delivered only once.

MQTT is a bi-directional communication protocol, where the clients can produce and consume data by publishing messages and subscribing to topics. Two-way communication enables IoT devices to send sensor data and simultaneously receive

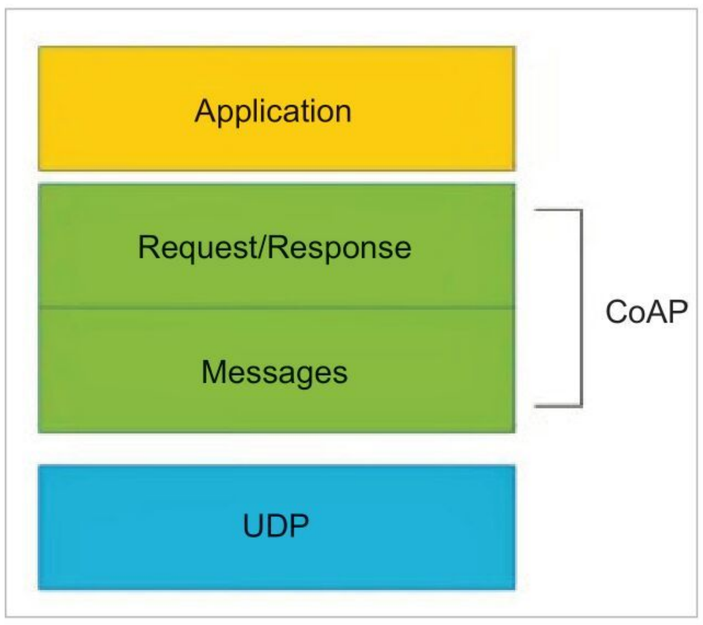


Figure 4: CoAP messaging model (Reference: *https://dzone.com/articles/coap-protocol-step-by-step-guide)*

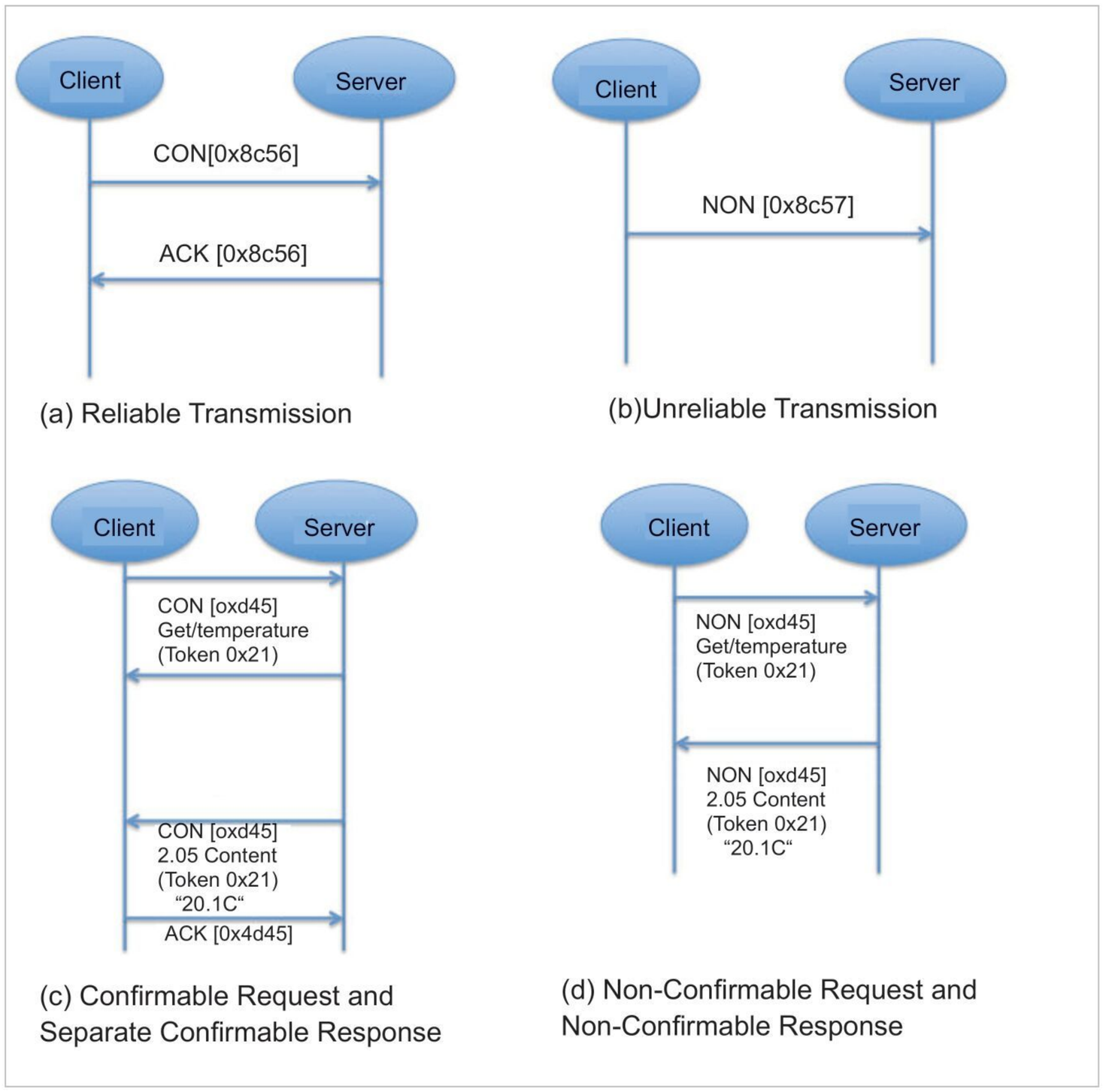


Figure 5: CoAP message flows *(https://devopedia.org/constrained-application-protocol)*

configuration information and control commands. Through MQTT, encrypting messages using TLS and verifying clients using modern authentication protocols becomes much easier.

Constrained application protocol (CoAP)

CoAP is a Web transfer protocol for constrained devices and networks in IoT. It can be implemented over a user datagram protocol (UDP), and is designed for applications with limited capacity to connect using light-weight machine-to-machine (LWM2M) communication — such as smart energy and building automation. LWM2M allows remote management of IoT devices, and provides interfaces to securely monitor and regulate them. CoAP architecture is based on the famous REST model, where servers make resources available under a URL, and clients access these resources using methods such as GET, PUT, POST, and DELETE. The CoAP and HTTP protocols have many similarities, with the difference that CoAP is enhanced for IoT, and more specifically for M2M. It has low overhead, along with proxy and caching capabilities, and exchanges messages asynchronously. The architecture of CoAP is divided into two main categories: messaging, which is responsible for the reliability and duplication of messages; and request/ response, which is responsible for communication.

The message layer is on top of the UDP, and is responsible for exchanging messages between the IoT devices and Internet. CoAP has four different types of messages — confirmable, non-confirmable, acknowledgment and reset. A confirmable message (CON) is a reliable message when exchanged between two endpoints. It is sent over and over again until the other end sends an acknowledge message (ACK). The ACK message has the same message ID as that of a CON. If the server faces issues managing the incoming request, it can send back a reset message (RST) instead of an ACK. For exchanging non-critical messages, unreliable NON messages can be used, where the server does not acknowledge the message. NON messages are assigned message IDs to detect duplicate messages.

Request/response is the second layer of the CoAP abstraction layer, which uses CON or NON messages to send requests. In scenarios where a server can respond immediately, a request is sent using a CON message, with an ACK message containing the response, or the error code is sent back by the server. The request and the response both have the same token, different from the message ID. In places where the server cannot respond immediately, it sends an ACK message with no content as the response. Once the response is ready, a new CON message containing the response is sent back to the client and the client acknowledges the response received.

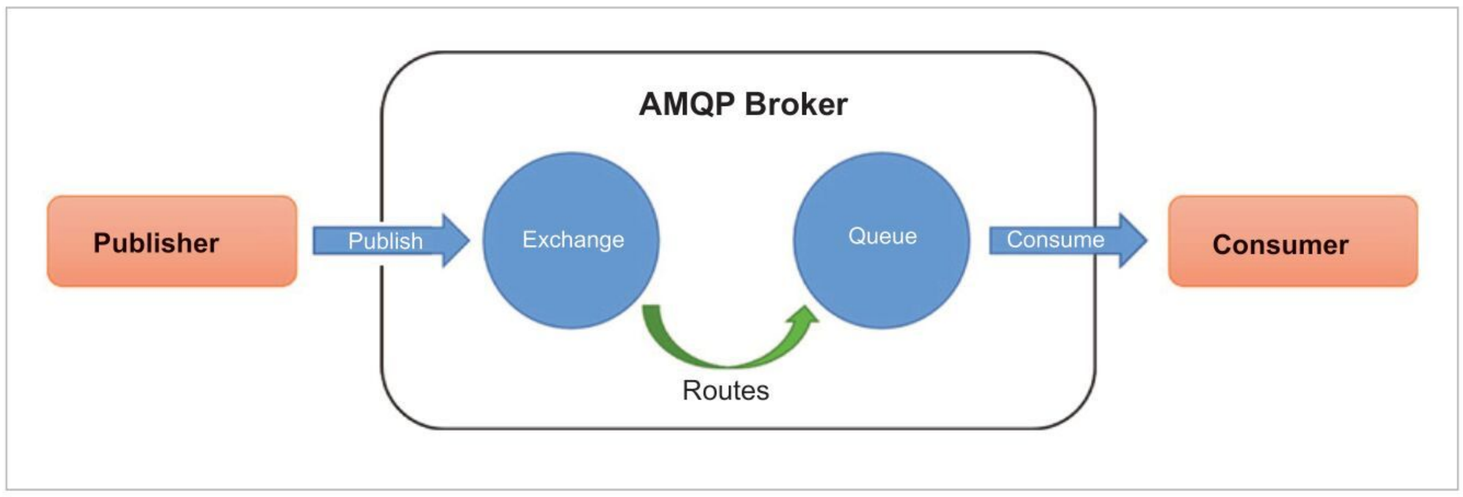


Figure 6: AMQP architecture (Reference: *https://www.researchgate.net/figure/Advanced-message-queueing-protocol-architecture_fig8_305987854)*

Advanced message queuing protocol (AMQP)

AMQP is an open standard application layer protocol designed for higher security, reliability, easy provisioning, and interoperability. It is a connection-oriented protocol, which means the client and the broker need to establish a connection before they transfer data because TCP is used as a transport protocol. AMQP offers two levels of QoS for reliable message delivery — unsettle format (similar to MQTT QoS0) and settle format (similar to MQTT QoS1). The main difference between AMQP and MQTT standards is that in AMQP the broker is divided into two main components — exchange and queues. Exchange is responsible for receiving publisher messages and sending them to queues. Subscribers connect to those queues, which basically represent the topics, and receive the sensory data whenever they are available.

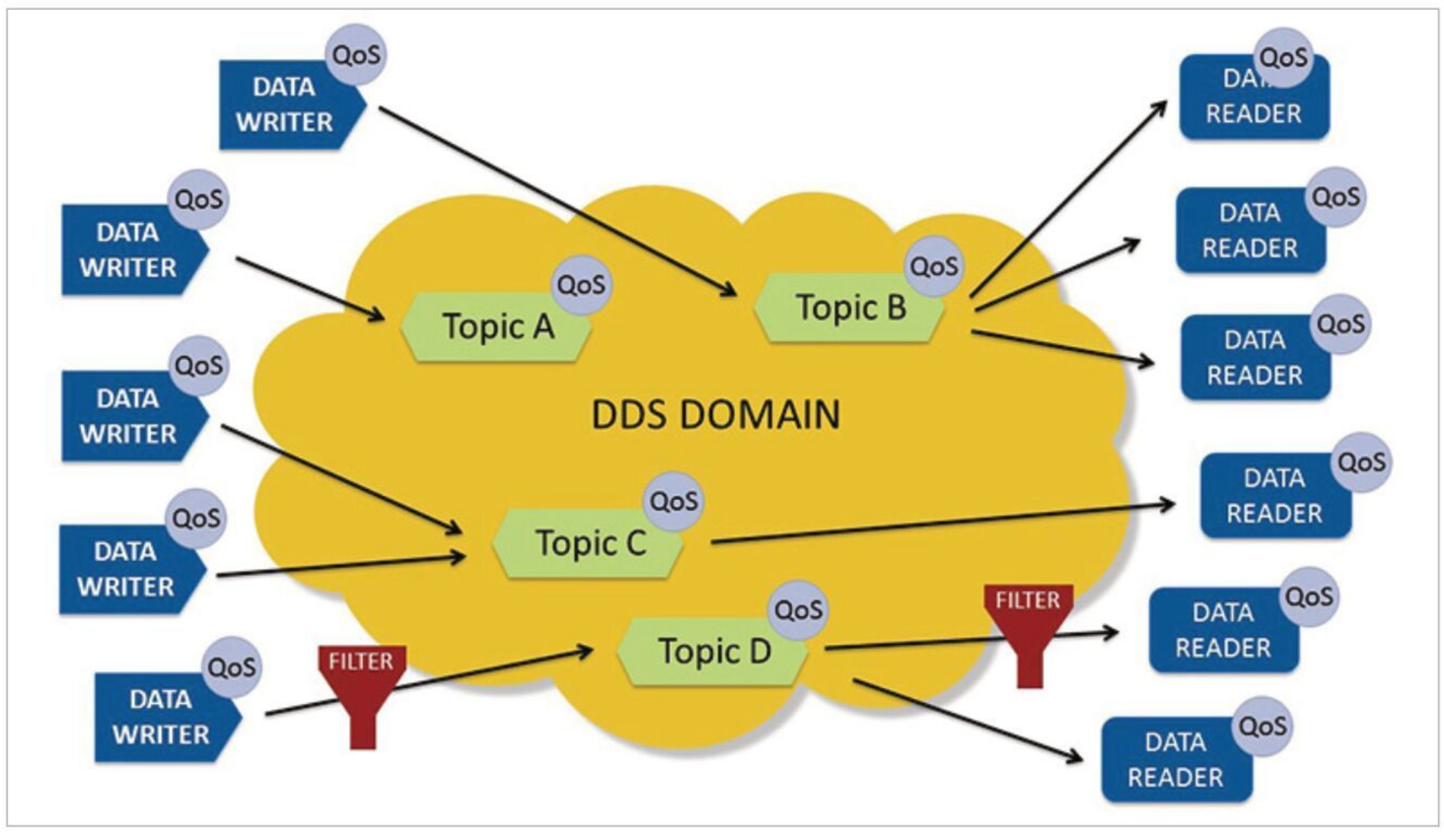


Figure 7: DDS data-centric model (Reference: *https://www.dds-foundation.org/what-is-dds-3//*

Table 1: A comparative analysis of IoT protocols

Features	CoAP	MQTT	AMQP	DDS
Messaging pattern	request/response	publish-subscribe	request/response; publish-subscribe	publish-subscribe
Architecture	tree	tree	star	bus
Transport	UDP	TCP, MQTT-S: UDP	TCP	UDP or TCP
Network layer	IPv6	IPv4 or IPv6	IPv4 or IPv6	IPv6
QoS level	2 levels	3 levels	3 levels	23 levels
Communication scope	Device-to-device	Device-to-cloud	Device-to-device, Device-to-cloud, Cloud-to-cloud	Device-to-device, Device-to-cloud, Cloud-to-cloud
Addressing	URI	topic only	queue, topic/Routing Key	topic/key
Security	DTLS, IPsec	TLS	SASL/TLS	TLS, DTLS, DDS security
Interoperability	Semantic	Foundational	Structural	Semantic
[Broker/server] implementation language	Java, C, C#, C++, Erlang, Go, Python, JavaScript, Ruby, Rust, Swift	Java, C, C#, C++, JavaScript, Erlang, Go, Lua, Python	Java, C, C#, C++, Python, Ruby	Ada, Java, C, C#, C++, Python, Scala, Lua, Pharo, Ruby
Header size	4 bytes	2 bytes	8 bytes	16 bytes
Governing body	IETF	OASIS	OASIS	OMG

Data Distribution Service (DDS)

DDS is a middleware protocol for data-centric connectivity from the object management group, which provides low-latency, data connectivity, extreme reliability, and a scalable architecture for business and mission-critical IoT applications. This protocol supports multicasting techniques in data transmission and high-quality QoS in small memory footprint devices and applications. The messaging model in DDS consists of two interface layers — data-centric, publish-subscribe (DCPS) and data local, reconstruction layer (DLRL). The DCPS layer is responsible for binding the values of data objects within an application during the publish/subscribe process. The DLRL is an optional layer used for integrating DDS at the application level.

There are two main constructs in DCPS — publisher with DataWriter and subscriber with DataReader. A publisher uses the DataWriter to bind values of data objects for each defined data type. It is responsible for data distribution while adhering to the QoS policies, if any. The DataWriter describes the data that requires publishing when an application needs to write data associated with data objects. A subscriber receives published data, based on a predefined set of QoS policies, from an application. An application uses a DataReader attached to a subscriber to retrieve the data from the latte. It subscribes to data described by a DataReader, which is provided by a known subscriber. Publishing and subscribing to data objects is done by using topics. A topic is associated with a name, data type and one or more QoS policies related to the data. The QoS policies are often used to manage how the data is distributed in the messaging model. DDS provides 23 different QoS levels with features such as security, durability, reliability, and many others. Figure 7 presents a high-level overview of the communication model of the DCPS layer in DDS.

Basic similarities among the several IoT messaging protocols that exist today (Table 1) suggest that they have the potential to coexist throughout the design and deployment of IoT systems. Given the diversity among IoT device types and the protocols they can support, it is not rare for IoT systems to employ several protocols for IoT data exchange. Since messaging protocols are key components for the connectivity of IoT devices, understanding their strengths and weaknesses is important while choosing an appropriate one. END

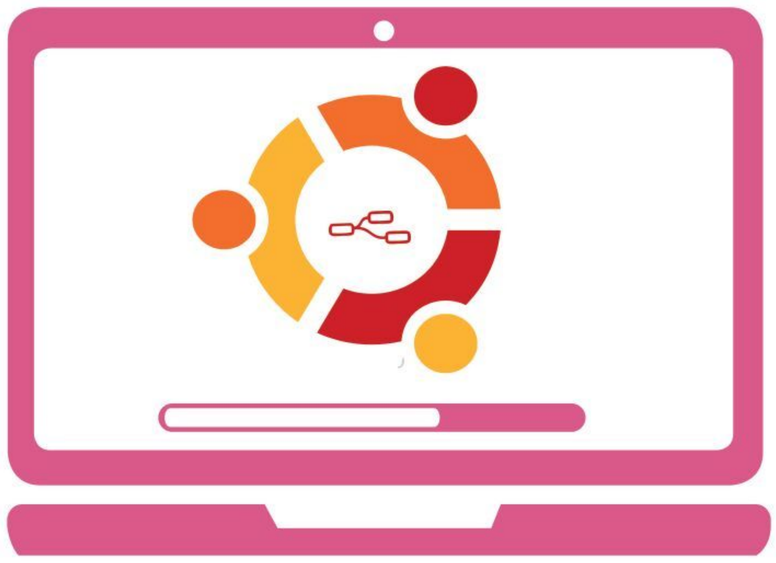
References

[1] IoT architecture *(https://www.hindawi.com/journals/jece/2017/9324035/)*
[2] Types of IoT connections *(https://www.sam-solutions.com/blog/internet-of-things-iot-protocols-and-connectivity-options-an-overview/)*
[3] MQTT publish/subscribe architecture *(https://mqtt.org/)*
[4] CoAP messaging model *(https://dzone.com/articles/coap-protocol-step-by-step-guide)*
[5] CoAP message flows *(https://devopedia.org/constrained-application-protocol)*
[6] AMQP architecture *(https://www.researchgate.net/figure/Advanced-message-queueing-protocol-architecture_fig8_305987854)*
[7] DDS data-centric model *(https://www.dds-foundation.org/what-is-dds-3/)*

By: Divya Singh

The author is a networking geek with eight years of experience in the IT industry as a domain network consultant. She has an insatiable curiosity for DevOps and cloud. She loves travelling, reading, and writing on social issues and travel experiences. She is an aspiring best-selling author.

Installing and Using Node-RED on Ubuntu



Node-RED adds to the benefits of Node.js. This article shows how to install and use it on the popular open source operating system Ubuntu.

Node-RED is a hardware device wiring tool that uses programming. Different nodes of devices, hardware and other APIs can be wired together in a browser based tool using Node-RED. It has become very important software in the field of IoT as it connects APIs, hardware devices and online services in new and interesting ways. This lightweight tool has been built on Node.js because of the amazing non-blocking and event-driven model it provides. It can be said that Node-RED basically acts as a controller for IoT devices. Hence, it is advisable for people interested in IoT to learn more about this software.

The first step is to install Node.js, since Node-RED is developed on it. We do that by using the following command, as shown in Figure 1.

```
sudo apt-get install nodejs-legacy
```

Next, to check if our installation has taken place properly, we use the following command, as shown in Figure 2.

```
Node -v
```

```
jishnusaurav@jishnusaurav-IdeaPad-L340-15IRH-Gaming:~$ sudo apt-get install nodejs-legacy
[sudo] password for jishnusaurav:
Reading package lists... Done
Building dependency tree
Reading state information... Done
Note, selecting 'nodejs' instead of 'nodejs-legacy'
The following packages were automatically installed and are no longer required:
  libevent-core-2.1-6 linux-hwe-5.4-headers-5.4.0-73
  linux-hwe-5.4-headers-5.4.0-74 linux-hwe-5.4-headers-5.4.0-77
  linux-hwe-5.4-headers-5.4.0-80 linux-hwe-5.4-headers-5.4.0-81
  linux-hwe-5.4-headers-5.4.0-84 linux-hwe-5.4-headers-5.4.0-86
  linux-hwe-5.4-headers-5.4.0-87
Use 'sudo apt autoremove' to remove them.
The following packages will be upgraded:
  nodejs
1 upgraded, 0 newly installed, 0 to remove and 81 not upgraded.
Need to get 25.1 MB of archives.
After this operation, 1,289 kB of additional disk space will be used.
Get:1 https://deb.nodesource.com/node_14.x bionic/main amd64 nodejs amd64 14.18.1-1nodesource1 [25.1 MB]
Fetched 25.1 MB in 15s (1,672 kB/s)
N: Ignoring file 'nvi' in directory '/etc/apt/sources.list.d/' as it has no filename extension
(Reading database ... 579459 files and directories currently installed.)
Preparing to unpack .../nodejs_14.18.1-1nodesource1_amd64.deb ...
Detected old npm client, removing...
Unpacking nodejs (14.18.1-1nodesource1) over (14.17.1-1nodesource1) ...
Setting up nodejs (14.18.1-1nodesource1) ...
Processing triggers for man-db (2.8.3-2ubuntu0.1) ...
N: Ignoring file 'nvi' in directory '/etc/apt/sources.list.d/' as it has no filename extension
```

Figure 1: Node.js installation

```
jishnusaurav@jishnusaurav-IdeaPad-L340-15IRH-Gaming:~$ npm -v
6.14.5
jishnusaurav@jishnusaurav-IdeaPad-L340-15IRH-Gaming:~$
```

Figure 2: Checking the node version

```
jishnusaurav@jishnusaurav-IdeaPad-L340-15IRH-Gaming:~$ sudo npm install -g --unsafe-perm node-red
 node-red-admin


added 351 packages, and audited 351 packages in 28s

28 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
npm notice
npm notice New major version of npm available! 7.0.14 -> 8.1.4
npm notice Changelog: https://github.com/npm/cli/releases/tag/v8.1.4
npm notice Run npm install -g npm@8.1.4 to update!
npm notice
jishnusaurav@jishnusaurav-IdeaPad-L340-15IRH-Gaming:~$
```

Figure 3: *node-red-admin* installation

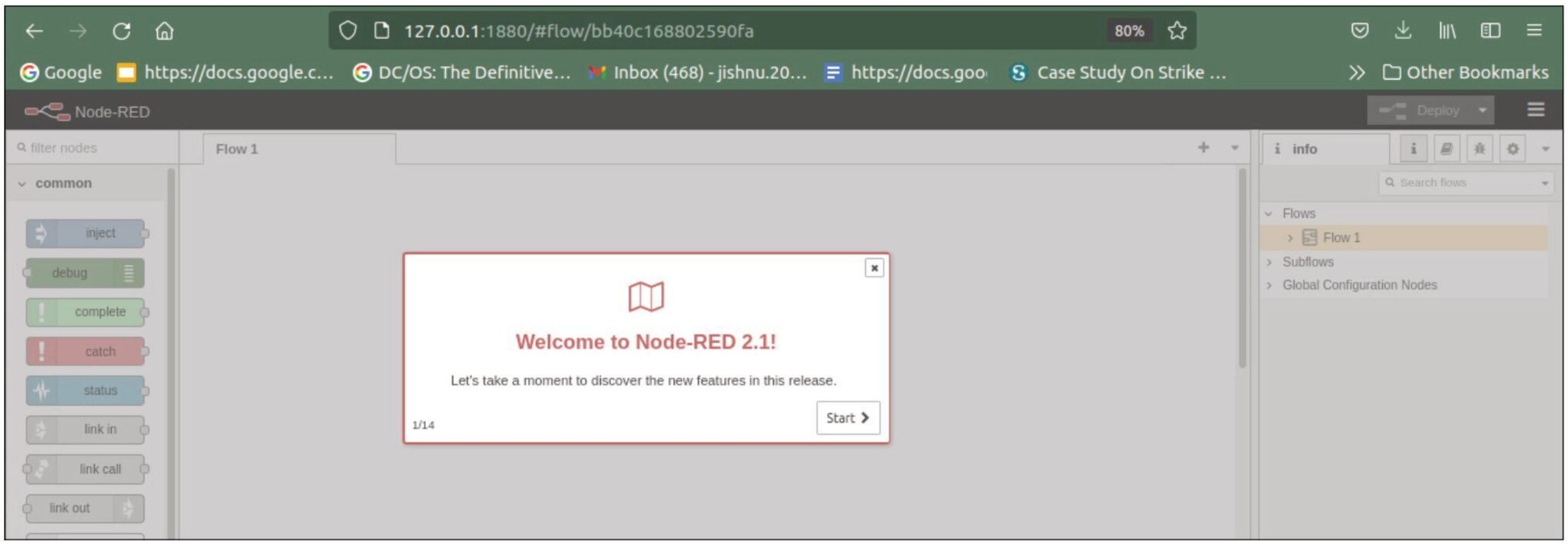


Figure 4: Node-RED in the browser

```
jishnusaurav@jishnusaurav-IdeaPad-L340-15IRH-Gaming:~$ node-red
23 Nov 23:08:31 - [info]

Welcome to Node-RED
===================

23 Nov 23:08:31 - [info] Node-RED version: v2.1.3
23 Nov 23:08:31 - [info] Node.js  version: v14.4.0
23 Nov 23:08:31 - [info] Linux 5.4.0-90-generic x64 LE
23 Nov 23:08:31 - [info] Loading palette nodes
23 Nov 23:08:32 - [info] Settings file  : /home/jishnusaurav/.node-red/settings.js
23 Nov 23:08:32 - [info] Context store  : 'default' [module=memory]
23 Nov 23:08:32 - [info] User directory : /home/jishnusaurav/.node-red
23 Nov 23:08:32 - [warn] Projects disabled : editorTheme.projects.enabled=false
23 Nov 23:08:32 - [info] Flows file     : /home/jishnusaurav/.node-red/flows.json
23 Nov 23:08:32 - [info] Creating new flow file
23 Nov 23:08:32 - [warn]

---------------------------------------------------------------------
Your flow credentials file is encrypted using a system-generated key.

If the system-generated key is lost for any reason, your credentials
file will not be recoverable, you will have to delete it and re-enter
your credentials.

You should set your own key using the 'credentialSecret' option in
your settings file. Node-RED will then re-encrypt your credentials
file using your chosen key the next time you deploy a change.
---------------------------------------------------------------------

23 Nov 23:08:32 - [info] Server now running at http://127.0.0.1:1880/
23 Nov 23:08:32 - [info] Starting flows
23 Nov 23:08:32 - [info] Started flows
```

Figure 5: Adding the script

Now, we need to install NPM (node package manager).

```
sudo apt-get install npm
```

To check the installation, use the following command:

```
npm -v
```

The next step is to install a helping utility called *node-red-admin*, using the following function, as shown in Figure 3.

```
sudo npm install -g --unsafe-perm node-
red node-red-admin
```

We now need to allow one of our firewall ports. Since 1880 is a comparatively less used port, let us proceed using the following command:

```
sudo ufw allow 1880
```

Finally, we can start Node-RED by just entering it on the terminal, as shown in Figure 4. There will now be a link out, which probably is *http://127.0.0.1:1880/*. We can now use Node-RED in the browser, as shown in Figure 4.

A small script needs to be added to start Node-RED automatically on startup. Use the following command for that:

```
sudo nano /etc/systemd/system/node-red.
service
```

Once the editor of the *.service* file opens up, enter the following script, as shown in Figure 5.

```
[Unit]
Description=Node-RED
After=syslog.target network.target

[Service]
ExecStart=/usr/local/bin/node-red-pi
--max-old-space-size=128 -v
Restart=on-failure
KillSignal=SIGINT

# log output to syslog as ‘node-red’
SyslogIdentifier=node-red
StandardOutput=syslog

# non-root user to run as
WorkingDirectory=/home/sammy/
User=sammy
Group=sammy

[Install]
WantedBy=multi-user.target
```

To check if our Node-RED is still working properly, we can use the following command:

```
sudo systemctl start node-red
```

If there are no errors, it means that everything has been set up perfectly.

```
    //    * The function should return true if the connection should be accepted, false other
    //    *
    //    * Alternatively, if this function is defined to accept a second argument, callback,
    //    * it can be used to verify the client asynchronously.
    //    * The callback takes three arguments:
    //    *   - result : boolean, whether to accept the connection or not
    //    *   - code : if result is false, the HTTP error status to return
    //    *   - reason: if result is false, the HTTP reason string to return
    //    */
    //},
}

adminAuth: {
    type: "credentials",
    users: [{
        username: "admin",
        password: "$2b$08$NIJKvK2JFfV4lKau2u54f.8OBnHewV73C4Py4mnPhNTy4iskSCgu6",
        permissions: "*"
    }]
},
```

Figure 6: Setting a password

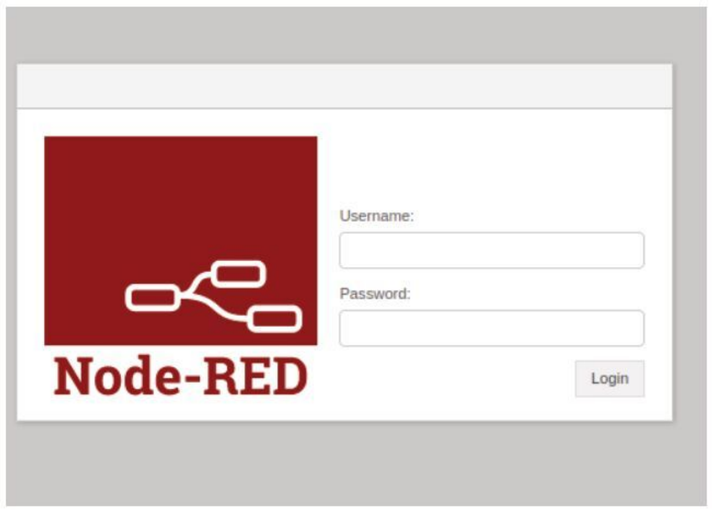


Figure 7: Final look after setting up the password

The final step is to provide at least some basic security to our machine, for which we need to run the following command:

```
node-red-admin hash-pw
```

This generates a hash value that you need to copy and keep.

Enter the following command to add the script, and then enter your password:

```
nano ~/.node-red/settings.js
```

Next, add the script shown in Figure 6 to secure your system. In this script, replace ‘passwd’ by the hash value. Our installation is now complete. If we go to our local host, it will ask for a password. Once the password is entered, we can use the tool normally as per our needs, as shown in Figure 7.

If you have liked this short tutorial, do try it out to connect your hardware devices. END

By: Jishnu Saurav Mittapalli

The author is a full stack Web developer, interested in modern technologies like artificial intelligence and machine learning. He is currently working in the field of natural language processing.

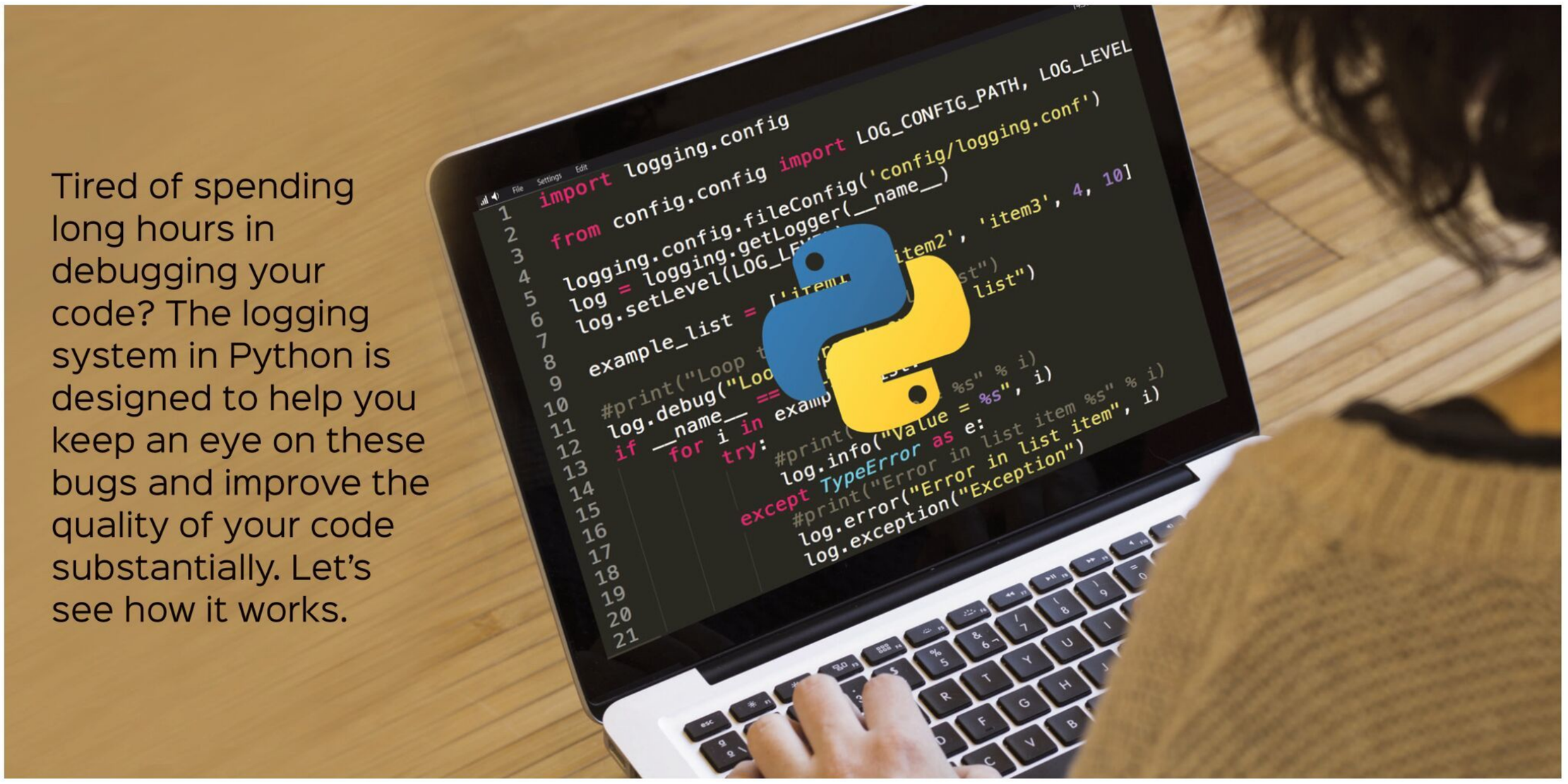
THE COMPLETE MAGAZINE ON OPEN SOURCE

» **To find dealers** to buy copy from news-stand, **visit: www.ezine.lfymag.com**

» **To subscribe**, **visit: pay.efyindia.com**

» **To buy an ezine edition**, **visit: www.magzter.com** choose **Open Source For You**

Want to Write Quality Code? Try Logging in Python



Tired of spending long hours in debugging your code? The logging system in Python is designed to help you keep an eye on these bugs and improve the quality of your code substantially. Let's see how it works.

Logging is an important part of software development and helps reduce endless hours of debugging. It can be used for different flow analyses, health checks, issue analysis, performance checks, security scans, audit and so on, if used correctly.

Log calls from the program indicate that certain events have occurred during the timestamp, may be from the IP address, and also give some additional information. The events hold the descriptive message — for example, dynamic or static values, importance of message (debug, info, warn, error), etc.

Python provides a logging system as a part of its standard library, making it easier for you to add it to your application.

Logging module

The logging module that helps add logs is easy and readily available, as shown below:

```
import logging

logging.debug('This is a debug message')
logging.info('This is an info message')
logging.warning('This is a warning message')
logging.error('This is an error message')
logging.critical('This is a critical message')
```

The output is:

```
WARNING:root:This is a warning message
ERROR:root:This is an error message
CRITICAL:root:This is a critical message
```

As seen from the code, there are five levels of logging, which indicate the severity of events. Each level has a corresponding severity associated. The levels defined in Table 1 are based on the severity in increasing order.

Level	Used for
Debug	Detailed messages, usually for diagnosing problems
Info	Confirmation or facts of working as expected
Warning	Alarming for future breaks, but until now the application is working
Error	Due to a serious problem in the function
Critical	Due to a serious problem, the program failed to continue to execute

Table 1: Usage of each log level

From the output, it can be seen that the debug and info messages are not being logged, as the default severity is set to 'warning'. So messages will be printed for the warning level and above.

Let us now see how the severity levels can be changed by changing the configurations. We will also look at how to change the format of the output, later on.

Parameter	Description
level	Root logger will be set to the specified severity level
filename	Name of the file logs should be stored
filemode	Mode in which the log file should be opened; default is 'a' append mode
format	Pattern for the log message to be written
datefmt	Format in which date should be written

Table 2: Commonly used log parameters

Several configurations can be done.

Basic configuration

We can use the method given below to configure the logging:

```
import logging
logging.basicConfig(level=logging.DEBUG, filename='/project/
log/app.log', filemode='w', format='%(asctime)s - %(message)
s', datefmt='%d-%b-%y %H:%M:%S')

logging.debug('This is a debug message')
logging.info('This is an info message')
logging.warning('This is a warning message')
logging.error('This is an error message')
logging.critical('This is a critical message')
```

The output (from *app.log*) is:

```
27-Nov-21 07:28:09 - This is a debug message
27-Nov-21 07:28:09 - This is an info message
27-Nov-21 07:28:09 - This is a warning message
27-Nov-21 07:28:09 - This is an error message
27-Nov-21 07:28:09 - This is a critical message
```

Through this basic configuration we observe that:

- Debug and info messages are logged, as the logging level is now 'debug' and above.
- Log contents are not displayed in *stdout;* rather, they are saved in the file called *app.log*.
- Message format is different, because it works as per the format given in the format parameter.
- The date format is also different.

File configuration

Before we get to know the file configuration, let's look at how logging works.

Logging flow: So far, we have seen the default logger named root, which is used by the logging module whenever its functions are called directly, like *logging.debug()*. But it is preferable to define your own logger.

The logging module follows the modular approach that splits functionalities into components.

1. **Logger:** This is exposed to application code for direct usage, and the logging is performed by calling methods on instances of the Logger class. This class performs three duties.

- Exposes several methods to application code so that applications can log messages at runtime. A good convention is to use the module name to name the loggers.

  - `logger = logging.getLogger(__name__)`

- Logger objects determine which log messages to act upon based upon the severity or filter objects.

  - `Logger.setLevel()`
  - `Logger.addFilter()and Logger.removeFilter()`

- Logger objects pass along relevant log messages to all interested log handlers.

  - `Logger.addFilter()and Logger.removeFilter()`

2. **LogRecord:** Loggers automatically create LogRecord objects that have all the information related to the event being logged, like the name of the logger, function, line number, message, and more.
3. **Filter:** This provides a finer grained facility for determining which log records to output. We will later see an example that uses filters.
4. **Handler:** This sends the LogRecords (created by loggers) to the appropriate destination. For example, it could be file, stdout, http, etc. For each destination, handler has a subclass such as file: FileHandler, stdout: StreamHandler, http: HTTPHandler. Logger objects can add zero or more handler objects to themselves with an *addHandler()* method.
5. **Formatter:** This specifies the format of LogRecord by mentioning it in the list of attributes that need to be logged in order to be written in the final output.

We can configure logging in three ways:

1. By creating loggers, handlers, and formatters explicitly using Python code that calls the configuration methods. In the example given below, though a logger is created with the same module name, the console handler is created

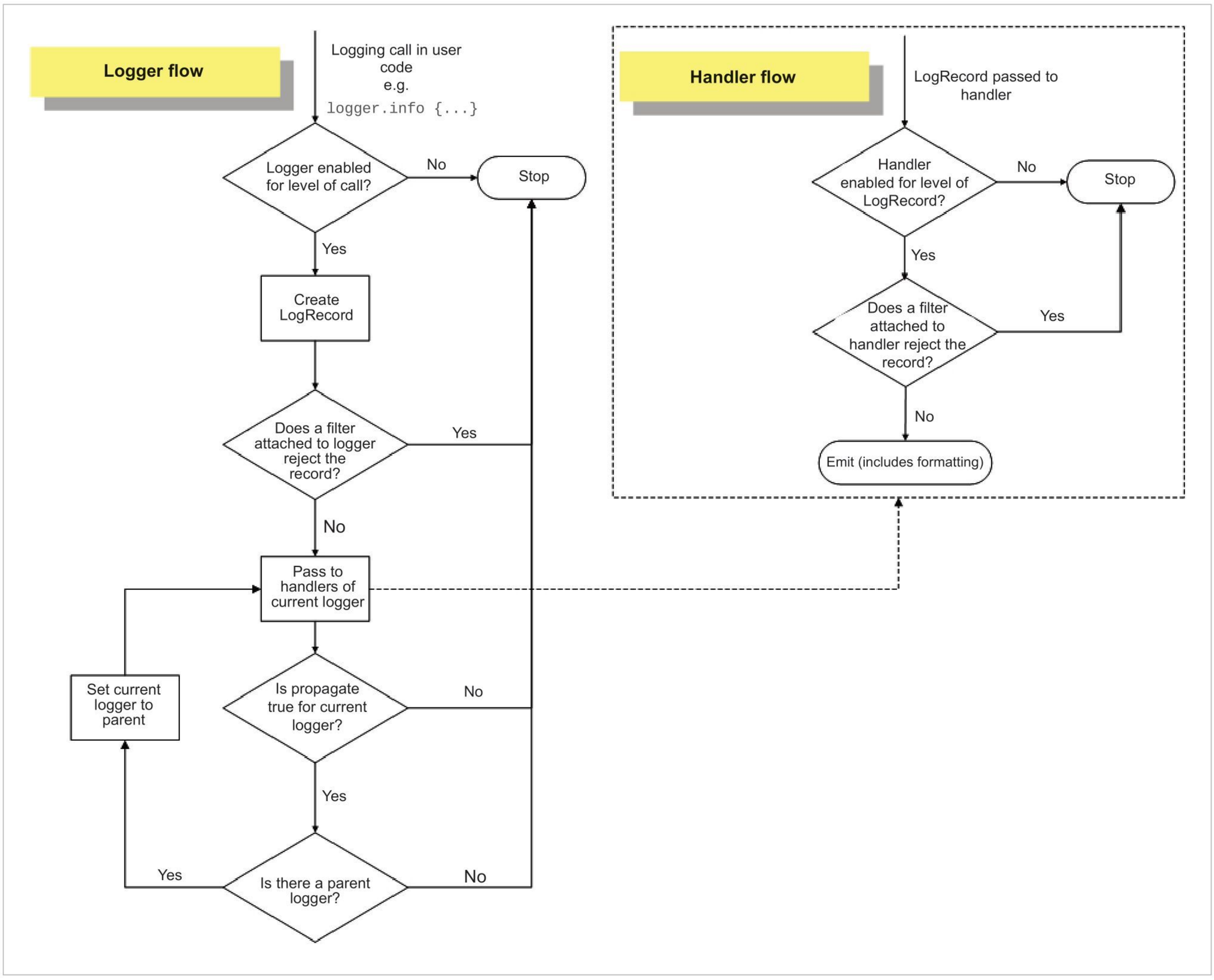


Figure 1: Logging flow (Source: *https://docs.python.org/3/howto/logging.html*)

with the debug level set and the formatter is created with time, module name, level and message.

```
import logging
# create logger
logger = logging.getLogger(__name__)
logger.setLevel(logging.DEBUG)
# create console handler and set level to debug
ch = logging.StreamHandler()
ch.setLevel(logging.DEBUG)
# create formatter
formatter = logging.Formatter('%(asctime)s - %(name)s -
%(levelname)s - %(message)s')
# add formatter to ch
ch.setFormatter(formatter)
# add ch to logger
logger.addHandler(ch)
# 'application' code
```

```
logger.debug('debug message')
logger.info('info message')
logger.warning('warn message')
logger.error('error message')
logger.critical('critical message')
```

The output is:

```
$ python handlerconfig.py

2021-11-28 14:45:45,096 - __main__ - DEBUG - debug message
2021-11-28 14:45:45,097 - __main__ - INFO - info message
2021-11-28 14:45:45,097 - __main__ - WARNING - warn message
2021-11-28 14:45:45,097 - __main__ - ERROR - error message
2021-11-28 14:45:45,097 -__main__- CRITICAL- critical message
```

2. By creating a logging *config* file and reading it using the *fileConfig()*function.

Shown below is an example where a logger is created with the 'simpleExample' logger name, a console handler with the debug level set, and file handler with info level set, making sure that the debug log will not be printed. Note that both the configurations are taken from the *logging.conf* file and it's important to have the handler and logger set for 'root'.

- In file handler:
  - The *propagate* entry is set to 1 to indicate that messages must propagate to handlers higher up the logger hierarchy from this logger, or 0 to indicate that messages are not propagated to handlers up the hierarchy. The *qualname* entry is the hierarchical channel name of the logger – it is the name used by the application to get the logger.
- In file rotate handler:
  - Here we have used file rotate handler to avoid logging all the content in one huge file. We have configured maxBytes=10, which means about 10 bytes written will be rotated. backupcount=5 indicates how many rotated files need to be kept, and then the remaining are cleaned.

3. By creating formatter with time, module name, level and message:

fileconfig.py

```
import logging
import logging.config
logging.config.fileConfig('logging.conf')
# create logger
logger1 = logging.getLogger('simpleExample')
logger = logging.getLogger()
# 'application' code
logger.debug('debug message')
logger1.info('info message')
logger1.warning('warn message')
logger1.error('error message')
logger1.critical('critical message')
```

loggin.conf

```
[loggers]
keys=root,simpleExample
[handlers]
keys=consoleHandler,simpleFileHandler
[formatters]
keys=simpleFormatter
[logger_root]
level=DEBUG
handlers=consoleHandler
[logger_simpleExample]
level=INFO
handlers=simpleFileHandler
qualname=simpleExample
propagate=0
[handler_consoleHandler]
class=StreamHandler
formatter=simpleFormatter
args=(sys.stdout,)
[handler_simpleFileHandler]
class=handlers.RotatingFileHandler
formatter=simpleFormatter
maxBytes=10
args=('test.log','a',10,5)
[formatter_simpleFormatter]
format=%(asctime)s - %(name)s - %(levelname)s - %(message)s
```

The output is:

```
$ python fileconfig.py

2021-11-28 17:47:26,858 - root - DEBUG - debug message
$ tail test.log*
==> test.log <==
2021-11-28 18:48:38,520-simpleExample - CRITICAL - critical
message
==> test.log.1 <==
2021-11-28 18:48:38,519 - simpleExample - ERROR - error
message
==> test.log.2 <==
2021-11-28 18:48:38,518 - simpleExample - WARNING - warn
message
==> test.log.3 <==
2021.11.28 8:48:38,517 - simpleExample - INFO - info message
```

In the next example, let's see how to create a dictionary of configuration information and pass it to the *dictConfig()* function. This is simple and can be done using the *yaml* format as configuration. It creates a logger with the same module name, a console handler with the debug level set and a formatter with time, module name, level and message. The *dictConfig* expects input to be in the *dict* object, so that *dict* can be defined and the logger method called.

dict_filelog.py

```
import logging
import logging.config
import yaml
with open('dict_config.yaml', 'r') as f:
    config = yaml.safe_load(f.read())
    logging.config.dictConfig(config)
logger = logging.getLogger(__name__)
logger.debug('This is a debug message')
logger.info('This is a info message')
logger.error('This is a error message')
```

```
dict_config.yaml

version: 1
formatters:
  simple:
    format: '%(asctime)s - %(name)s - %(levelname)s -
%(message)s'
handlers:
  console:
    class: logging.StreamHandler
    level: INFO
    formatter: simple
    stream: ext://sys.stdout
loggers:
  sampleLogger:
    level: DEBUG
    handlers: [console]
    propagate: no
root:
  level: DEBUG
  handlers: [console]

The output is:

2021-11-28 19:25:00,508 -_main_-DEBUG-This is a debug message
2021-11-28 19:25:00,508 -_main_-INFO - This is a info message
2021-11-28 19:25:00,509 -_main_-ERROR-This is a error message
```

Note: When no configuration is provided, the events become an output using a handler of last resort, which is stored in *logging.lastResort*. This acts as StreamHandler, which writes messages to *sys.stderr*. Handler's level is set to 'warning' and the behaviour of the logging package in these circumstances is dependent on the Python version. In this article we are using Python 3.2 and above.

Logging from multiple modules

- Calling the same *getLogger("NAME")* multiple times will return a reference to the same object. This can be used across modules to get the same object but requires the same Python interpreter process.
- The application code can define and configure a parent logger in one module, and can only create but not configure a child logger in a separate module. All logger calls to the child will pass up to the parent. The parent and child mapping is done by the name of the logger. Here is an example in the getLogger method, where all the configuration is inherited from the parent.

  - `parent: getLogger('test_app')`
  - `child: getLogger('test_app.sub_app')`

```
File: main_app.py

import sys
import logging
import sub_app
# Create a custom logger with "test_app"
logger = logging.getLogger("test_app")
logger.setLevel(logging.DEBUG)
# Create handlers
# create console handler with debug log level
c_handler = logging.StreamHandler(sys.stdout)
# create file handler with error log level
f_handler = logging.FileHandler('file.log', 'w')
f_handler.setLevel(logging.WARNING)
# Create formatters and add it to handlers
c_format = logging.Formatter('%(name)s - %(levelname)s -
%(message)s')
f_format = logging.Formatter('%(asctime)s - %(name)s -
%(levelname)s - %(message)s')
c_handler.setFormatter(c_format)
f_handler.setFormatter(f_format)
# Add handlers to the logger
logger.addHandler(c_handler)
logger.addHandler(f_handler)
print(c_handler)
print(logger)
logger.debug('This is a debug from main module')
logger.info('This is a info from main module')
logger.warning('This is a warning from main module')
logger.error('This is an error from main module')
logger.info("Before calling the sub_app_function")
sub_app.sub_app_function()
logger.info("After calling the sub_app_function")

File: sub_app.py

import logging
# create logger
module_logger = logging.getLogger('test_app.sub_app')
def sub_app_function():
    module_logger.error("Debug message from sub_app")
```

The output is:

```
% python multiple_modules_logger.py
<StreamHandler <stdout> (NOTSET)>
<Logger test_app (DEBUG)>
test_app - DEBUG - This is a debug from main module
test_app - INFO - This is a info from main module
test_app - WARNING - This is a warning from main module
test_app - ERROR - This is an error from main module
test_app - INFO - Before calling the sub_app_function
```

```
test_app.sub_app - ERROR - Debug message from sub_app
test_app - INFO - After calling the sub_app_function
% cat file.log
2021-11-28 23:26:04,332 - test_app - WARNING - This is a
warning from main module
2021-11-28 23:26:04,332 - test_app - ERROR - This is an error
from main module
2021-11-28 23:26:04,332 - test_app.sub_app - ERROR - Debug
message from sub_app
```

Logging from multiple threads

Logging from multiple threads is as simple as the normal one. To differentiate the logs generated from each thread, threadName can be used. An example of logging from the main thread and another thread is shown below.

```
import logging
import threading
import time
def worker(arg):
    while not arg['stop']:
        logging.debug('Hi from thread 2')
        time.sleep(0.5)
def main():
    logging.basicConfig(level=logging.DEBUG,
format='%(asctime)s - %(name)s - %(levelname)s -
%(threadName)s - %(message)s')
    info = {'stop': False}
    thread = threading.Thread(target=worker, args=(info,))
    thread.start()
    while True:
        try:
            logging.debug('Hello from main thread')
            time.sleep(1.0)
        except KeyboardInterrupt:
            info['stop'] = True
            break
    thread.join()
if __name__ == '__main__':
    main()
```

The output is:

```
% python threadlogger.py
2021-11-28 22:38:54,207 - root - DEBUG - Thread-1 - Hi from
thread 2
2021-11-28 22:38:54,207 - root - DEBUG - MainThread - Hello
from main thread
2021-11-28 22:38:54,712 - root - DEBUG - Thread-1 - Hi from
thread 2
2021-11-28 22:38:55,212 - root - DEBUG - MainThread - Hello
from main thread
2021-11-28 22:38:55,214 - root - DEBUG - Thread-1 - Hi from
thread 2
2021-11-28 22:38:55,716 - root - DEBUG - Thread-1 - Hi from
thread 2
2021-11-28 22:38:56,217 - root - DEBUG - MainThread - Hello
from main thread
2021-11-28 22:38:56,220 - root - DEBUG - Thread-1 - Hi from
thread 2
2021-11-28 22:38:56,725 - root - DEBUG - Thread-1 - Hi from
thread 2
2021-11-28 22:38:57,221 - root - DEBUG - MainThread - Hello
from main thread
2021-11-28 22:38:57,229 - root - DEBUG - Thread-1 - Hi from
thread 2
2021-11-28 22:38:57,734 - root - DEBUG - Thread-1 - Hi from
thread 2
2021-11-28 22:38:58,223 - root - DEBUG - MainThread - Hello
from main thread
2021-11-28 22:38:58,236 - root - DEBUG - Thread-1 - Hi from
thread 2
2021-11-28 22:38:58,738 - root - DEBUG - Thread-1 - Hi from
thread 2
2021-11-28 22:38:59,226 - root - DEBUG - MainThread - Hello
from main thread
2021-11-28 22:38:59,239 - root - DEBUG - Thread-1 - Hi from
thread
```

Context logging and filter

- ***LoggerAdapter:*** Sometimes the logging output is needed to contain contextual information in addition to the parameters passed to the logging call. For example, in a networked application, it may be desirable to log client-specific information in the log (e.g., remote client's user name, or IP address). An easy way in which you can pass contextual information to the output along with the logging event information is to use the *LoggerAdapter* class.
  In the example that follows we are adding the *host_ip* to all the log events.
- ***Filter:*** A filter can be used for both the logger and handler levels, as it can stop certain information from being passed. It will also inject additional information into the LogRecord, which will be logged. In our example below, we are using filters to allow only certain levels to be logged. Though the log level is set to 'debug', we are only allowing 'info' and 'error' to be logged.

Note: These are fictional examples, as use cases in real life vary.

Continued to page no...66

Managing Secrets with SecretsFoundry

While building an app, you need to manage environment variables and preserve their credentials. SecretsFoundry helps you to automatically fetch these variables from different secret managers, helping to make configurations across environments seamless.

One of the biggest challenges in application development is managing environment variables and preserving their credentials. Let's go over different approaches in detail. Imagine you are building a simple app that talks to a MongoDB database through a connection and starts serving on port 3000. As easy as it sounds, it is very wrong to hard-code credentials inside the code, as it will then run only on the specified system where it has been coded. Even if shared with fellow engineers, the code will not be able to connect with the MongoDB instance or run in different deployed environments.

```
server = Server()
mongoDB = new MongoDb("mongodb://username: password@
locahost:3000/testdb")
server.get('/users', function() {
mongoDB.getUsers()
})
server.start(port=3000)
```

While cross-environment configurations are pain points, following the 12 factor app methodology helps in separating them from the code. Your configuration comprises credentials that talk to the backend services (SQL, MongoDB or Memcached). There can also be credentials that talk to different APIs if the code is interacting with Twitter or Amazon S3 APIs. There can be host names or ports that can vary across deployments, but this should be a part of the configuration.

What is 12 factor app methodology?

- Separate configuration from code
- Application's configuration is everything that varies between deploys
  - Resource handles to the database, Memcached and other services
  - Credentials to external services such as Amazon S3 or Twitter
  - Per-deploy values such as the canonical host name for the deploy

Defining environment variables

Generally, defining environment variables is OS-dependent, but programming languages have now abstracted away such differences using development packages like Python's *dotenv (.env).* A *dotenv* file can manage environment variables across dev environments, allowing the developer to use locally managed configuration files. After abstracting the configuration of code to a different *env* file, the code loads all variables from an *env* file.

```
DB_NAME=test_db
DB_USER=username
DB_PASSWORD=password
DB_DOMAIN=localhost
DB_PORT=3000
```

Figure 1: Abstracting the configuration to a *dotenv* file

The code is modified and MongoDB now uses the environment variables to connect to itself, making the code run in different places with a custom *dotenv* file.

```
// Parse .env using a package like dotenv
loadVariableFromEnv()

server = Server()
mongoDB = new MongoDb(
  process.env.DB_USER,
  process.env.DB_PASSWORD,
  process.env.DB_HOST,
  process.env.DB_PORT,
  Process.env.DB_NAME
))
server.get('/users', function() {
mongoDB.getUsers()
})
server.start(port=process.env.PORT)
```

Figure 2: Modifying code to use *dotenv*

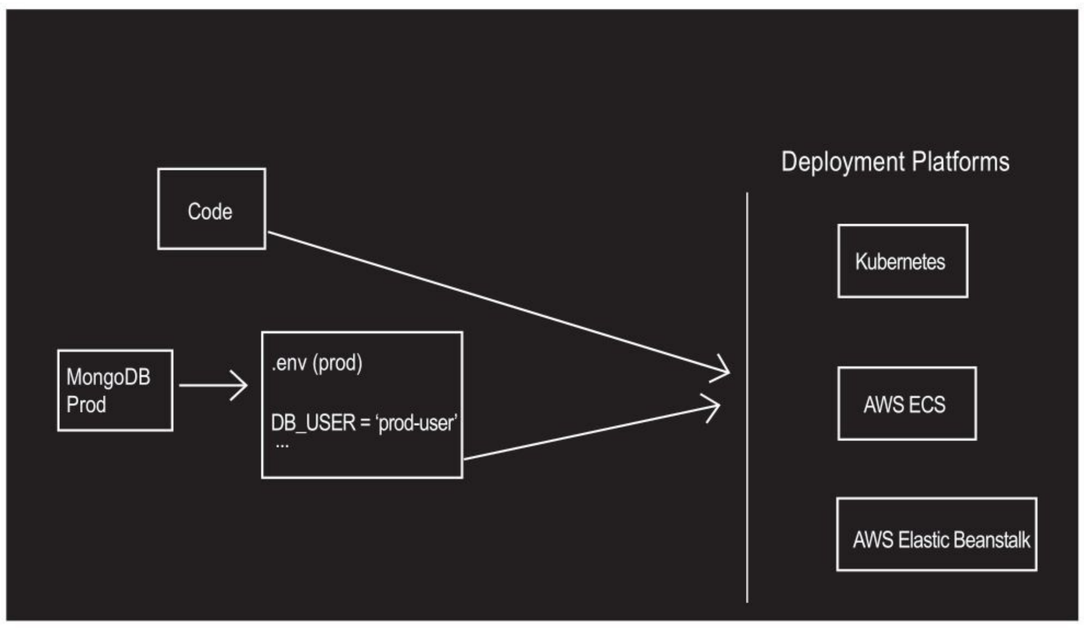


Figure 3: The variables for production environments are fed manually into the deployment infra

Deploying code to production

When the code is to be shifted to production, where you have a MongoDB production instance, you have to compose the *dotenv* file and copy the credentials of MongoDB to it. Similarly, you need to copy all the environment variables and send them to the deployment infrastructure. For example, the developer using Kubernetes will somehow have to set all the environment variables in it, and configure maps and secrets. For instance, an AWS ECS requires you to set that configuration in Fargate, and AWS Elastic Beanstalk has its own console for you to copy the environment variables.

The variables for production environments are fed manually to the deployment infra, which is awful. For every single app built, there is a manual process to copy variables into the deployment infrastructure. This is not a one-time process — every time a developer adds or deletes an environmental variable, it has to be further communicated to the DevOps team, and the latter either adds the variable or removes it from the production system. This is a slow manual cycle, and handling multiple apps with multiple environments

.env.development	.env.staging	.env.production
NODE_ENV=test	NODE_ENV=staging	NODE_ENV=production
DB_NAME=test_db	DB_NAME=staging_db	DB_NAME=prod_db
DB_USER=username	DB_USER=staging_username	DB_USER=prod_username
DB_PASSWORD=password	DB_PASSWORD=staging_password	DB_PASSWORD=prod_password
DB_DOMAIN=localhost	DB_DOMAIN=staging_url	DB_DOMAIN=prod_url
DB_PORT=3000	DB_PORT=5000	DB_PORT=3000
HOST=localhost	HOST=staging.example.com	HOST=example.com

Figure 4: Creating a *dotenv* file per environment

```
if (env is production) {
  config= loadFrom(.env.production)
} else if (env is staging) {
  config = loadFrom(.env.staging)
} else {
  config = loadFrom(.env.development)
}

server = Server()
mongoDB = new MongoDb(
  config.DB_USER,
  config.DB_PASSWORD,
  config.DB_HOST,
  config.env.DB_PORT,
  config.env.DB_NAME
))

server.get('/users', function() {
mongoDB.getUsers()
})

server.start(port=config.PORt)
```

Figure 5: Modifying code to get environment-specific configurations

becomes an extremely tedious and error-prone process.

A scalable solution is to create a *dotenv* file per environment for development, staging and production, as shown in Figure 4. We overwrite all the values and build a system that can just ship this file to the deployment infrastructure.

The whole process is now automated, and you just have to change the file and automatically sync to any deployment system that is being used. Once you have separate *dotenv* files, you can simply modify the code to load from whichever stage of production.

After getting the configuration, you can just call the MongoDB connection with config variables. The configuration should be written separately for every single environment and committed to the GitHub repository. Then, as the code is deployed using the continuous integration/continuous delivery (CI/CD) pipeline, it automatically loads the environment variables and runs, with no manual communication needed between the DevOps and engineering teams.

The *dotenv* file should be committed to Git and the environment variable must be set once in the deployment infrastructure manually. For the rest of the future deployments, the code will automatically run by passing on the variables.

The problem here is the need to put all these credentials for the production database in the *dotenv* production file and also commit it to Git. This is a huge security risk as you are sharing secret credentials with everyone in the project. There is the risk of any hacker getting access to the Git repository, even if you are working alone. Hence, fundamentally, never store credentials in plain text, or send them over Slack or any other messaging medium.

One solution here is to use a secrets manager, which will encrypt secrets and also have an access control built into it. So when the code asks for the encrypted key, the secrets manager automatically decrypts it and returns to the value. This allows you to manage permissions to people having access to the secrets, making the code more secure.

There are some secret managers that even provide active support for rotating secrets. For example, if the database credentials are stored in a secrets manager, the latter can automatically rotate these credentials every six months. So all the apps using the credentials from the secrets manager will start receiving the new ones after six months, and the database will also move to the new credentials. AWS Parameter Store, AWS Secrets Manager and cloud-agnostic solutions like Hashicorp Vault, GCP Secrets Engine and Azure Key Vault are a few examples of secrets managers that also make an app look simple.

You can leave *dotenv* files and move only sensitive parameters to a secrets manager. The code can be changed to read these variables from the two sources.

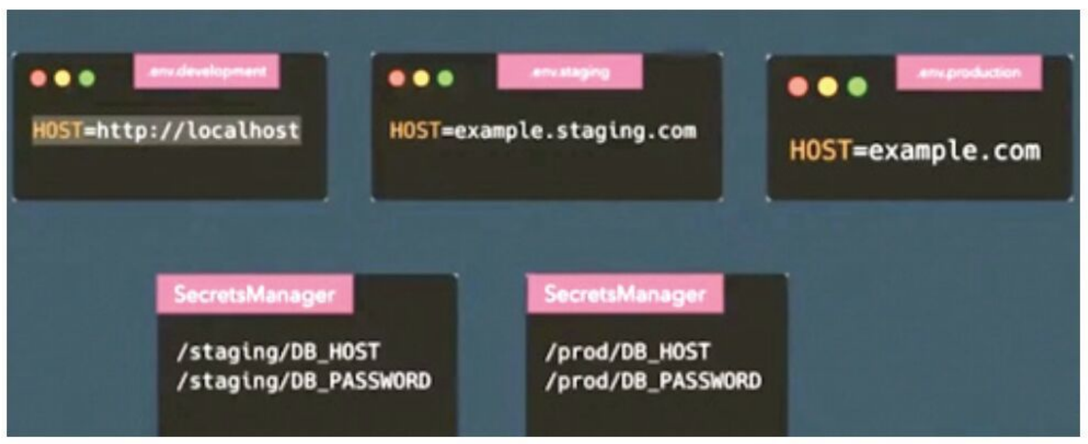


Figure 6: MongoDB credentials move to a secrets manager

```
config = loadFromEnv(.env.{stage})
if (production) {
  dbSecrets = getProdSecretsUsingApi()
} else if(staging) {
  dbSecrets = getStagingSecretsUsingApi()
} else {
  dbSecrets = getDevSecretsUsingApi()
}

server = Server()
mongoDB = new MongoDb(
  dbSecrets.DB_USER,
  dbSecrets.DB_PASSWORD,
  dbSecrets.DB_HOST,
  dbSecrets.DB_PORT,
  dbSecrets.NAME
))

server.get('/users', function() {
  mongoDB.getUsers()
})

server.start(port=config.PORT)
```

Figure 7: App code with secrets manager is not simple anymore

The code will now load things first from the *dotenv* file and then contact secrets manager API to get the actual value of the secrets. The API may vary accordingly, since the secrets are different for staging, production or development.

The code gets the secrets inside the application, and depending on which secrets manager is being used, the corresponding library is imported. This may not be a good solution, as there can be a lot of duplication of code for every single application inside the company. Abstracting it out as a library can make the code less feasible, and adding/deleting environments dynamically can be tedious.

Managing secrets via SecretsFoundry

Since there are major issues in moving the code to fetch secrets from inside the application and integrating it with multiple secrets managers, SecretsFoundry, a cloud-agnostic solution to make secrets management simple and secure, is of help. It figures out a solution to load all the secrets, passes it back to your application in a secure way, and also works on all platforms with different secrets managers.

Your code should now have a flow from *dotenv* files (production, staging or development) for each of the environments and then commit to Git. You are actually not

App.js

```
server = Server()
mongoDB = new MongoDb(
  process.env.DB_USER,
  process.env.DB_PASSWORD,
  process.env.DB_HOST,
  process.env.DB_PORT,
  Process.env.DB_NAME
))

server.get('/users', function() {
mongoDB.getUsers()
})

server.start(port=process.env.PORT)
```

Figure 8: Your app will now be secure

.env.development

```
DB_NAME=test_db
DB_USER=username
DB_PASSWORD=password
DB_DOMAIN=localhost
DB_PORT=3000
HOST=http://localhost
```

Figure 9: *Dotenv development* file

writing any of the credentials in these files, but just the path to the secrets manager. An example of code using Node.js is shown in Figure 8.

We are just accessing the *process dotenv* file directly without writing code to fetch any of the environment variables inside the code. However, the command to run the app needs to change.

```
secretsfoundry run --stage="development" -- script "node app.
js"
```

Behind the code, it looks at the environment variables. For example, if you see the *dotenv development* file, there is a database name, user name and password in plain text. It is safe in the local development environment and developers can access it using the *dotenv development.local* command. SecretsFoundry will read each of the variables and will put these as environment variables to run the application process. This may just be simple environment file parsing, but more critical changes happen in the production file.

```
DB_NAME=${aws-ssm:/production/db/name}
```

In the *dotenv production* file, the database name is written as *aws-ssm*, followed by the path to the secret. It tells your SecretsFoundry to fetch the value from the AWS Parameter Store with this key — *production/DB/name*, so as to give the actual value.

env.production

```
DB_NAME=${aws-ssm:/production/db/name}
DB_USER=${aws-secrets(region=us-east-2):/prod/db/username
DB_PASSWORD=${aws-secrets(region=us-east-2):/prod/db/password
DB_DOMAIN=${aws-ssm:/production/db/domain}
DB_PORT=3000
HOST=$PROD_HOST
```

Figure 10: *Dotenv production* file

```
DB_USER=${aws-secrets(region=us-east-2):/prod/db/username
```

In the DB user, the code should be written in a way that it can be fetched from the AWS Secrets Manager. In a similar way, we integrate with different providers like HashiCorp Vault and Amazon S3, so that you can just write in the *env* file where the source of the credential is, and SecretsFoundry will fetch it dynamically at runtime. This way it is much safer to commit the *dotenv production* file to Git.

How does SecretsFoundry work?

SecretsFoundry reads every variable defined in the *dotenv.** file according to the stage sent by the user in the *run* command and loads the necessary values.

Here is an example:

```
DB_PASSWORD=${aws-ssm;/app/staging/DB_PASSWORD}
```

You can see that the format of defining each environment (everything under the dollar symbol) is a variable that SecretsFoundry will try to parse out and first see the provider. The AWS Parameter Store is the provider here, with */app/staging/DB_PASSWORD* being the path to the secret inside it.

SecretsFoundry uses AWS APIs and secrets manager APIs to get values, and then puts in the environment variable to run the code. So your code will find the value of */app/staging/DB_PASSWORD* automatically in the environment variable *DB_password.*

This way, a developer can have as many *dotenv* files inside your repository and use the corresponding environment file by just passing this as an argument.

At the moment, SecretsFoundry can integrate only with AWS Parameter Store, AWS Secrets Manager, AWS S3 and HashiCorp Vault. It may soon be compatible with GCP Secrets Engine and Azure Key Vault. SecretsFoundry is currently available as an open source NPM package.

Container support

SecretsFoundry can also support containers and Kubernetes.

```
ENTRYPOINT ["secretsfoundry", "run", "--stage", "prod", "-s",
"node example.js"]
```

The developer working with the Docker file just has to change the *ENTRYPOINT* to *secretsfoundry* first and then *run* it, followed by the stage in which it is being run. Then the actual script, which was being written earlier as the *ENTRYPOINT*, must be included (as shown in the code snippet above).

This will execute the same logic when Docker starts. You also have to provide the access credentials for AWS Secrets Manager or for Hashicorp Vault once in the deployment infrastructure. And after that, developers can themselves keep pushing code without any manual intervention needed in the development environment for sinking environment variables.

You can also think of integrating SecretsFoundry into your CI/CD pipeline. When it runs, it will print the values of all the environment variables that it can parse out. END

This article is based on a talk given at OSI 2021 by Abhishek Choudhary, co-founder of Ensemble Labs (TrueFoundry).

References

[1] Open sourced code in GitHub: *https://github.com/truefoundry/secretsfoundry*
[2] Documentation: *https://truefoundry.gitbook.io/secretsfoundry/*

By: Abbinaya Kuzhanthaivel

The author is a technology journalist at *OSFY*.

Continued from page no...66

```
import sys
import logging
host_ip = "localhost"
class StdoutFilter(logging.Filter):
    def filter(self, record):
        return record.levelno in (logging.INFO, logging.
ERROR)
logger = logging.getLogger('sampleApp')
logger.setLevel(logging.DEBUG)
handler = logging.StreamHandler(sys.stdout)
handler.addFilter(StdoutFilter())
formatter = logging.Formatter('%(host_ip)s - %(asctime)s -
%(name)s - %(levelname)s - %(message)s')
handler.setFormatter(formatter)
logger.addHandler(handler)
logger = logging.LoggerAdapter(logger, {'host_ip': host_ip})
print(logger)
logger.debug('This is a debug from main module')
logger.info('This is a info from main module')
logger.warning('This is a warning from main module')
logger.error('This is an error from main module')
```

The output is:

```
<LoggerAdapter sampleApp (DEBUG)>
localhost - 2021-11-29 00:26:18,566 - sampleApp - INFO - This
is a info from main module
localhost - 2021-11-29 00:26:18,567 - sampleApp - ERROR -
This is an error from main module
```

Best practices

- Create loggers using the *getlogger* function
- Use appropriate log levels in different regions
- Create module level loggers
- Use rotating file handler
- Avoid opening the same log file multiple times
- Do not pass logger as parameter in class or methods
- Include timestamp in logs; it's preferable to use the standard format called ISO-8601. For example:

```
import logging
logging.basicConfig(format='%(asctime)s %(message)s')
logging.info('Example of logging with ISO-8601 timestamp')
```

As seen, the Python logging module is very flexible, customisable and easy to use. It has many more features such as context based logging, customisable log records, logging to multiple destinations and servers, multiple processes logging to the same file log, etc. The ready code is available in the Python cookbook URL at *https://docs.python.org/3/howto/logging-cookbook.html*.

If you haven't been using logging in your applications yet, it is a good time to begin doing so. You may be surprised at how much it enhances the quality of the code being written. END

References

[1] *https://realpython.com/python-logging/*
[2] *https://docs.python.org/3/howto/logging.html*
[3] *https://docs.python.org/3/howto/logging-cookbook.html*

By: Thangaselvi Arichandrapandian

The author works in a leading bank as AVP. She is a perennial learner and loves the quote: "The more I learn, the more I realise how much I don't know." - *Albert Einstein.*

Implementing Simple Linear Regression Using Python

In this article, let's look at the concept of linear regression with the help of an example and learn about its implementation in Python, and more.

Machine learning (ML) is a scientific study of algorithms and statistical models that computer systems use, in order to perform a specific task effectively without using explicit instructions.

Supervised learning is a type of ML in which a machine learns from labelled data and predicts the output from those learnings. Linear regression is a popular model that comes under supervised learning. Basically, regression algorithms are used for predicting continuous variables like stock prices, weather forecasts, etc.

What is linear regression?

Regression refers to the search for relationships among variables. We can relate it to the sales of ice-cream, which are determined by the weather of the city. With an increase in humidity, ice-cream sales also rise.

In simple terms, linear regression is used for finding the relationship between two variables — the input variable (x) and the output variable (y). A single output variable can be calculated from a linear combination of multiple input variables. Input variables are also known as independent variables or predictors, while output variables are also known as dependent variables or responses.

Let us look at the simplest algorithm used for prediction.

Learning model: Ordinary least squares (OLS)

This common method can have multiple inputs. The ordinary least squares procedure seeks to minimise the sum of the squared residuals. Each time, we calculate the distance from every data point to the regression line, square it, and take the sum of all the squared errors together, ending up with considering the parameters of the least sum of all squared errors. The following example elaborates this by predicting the price of wine, given its age.

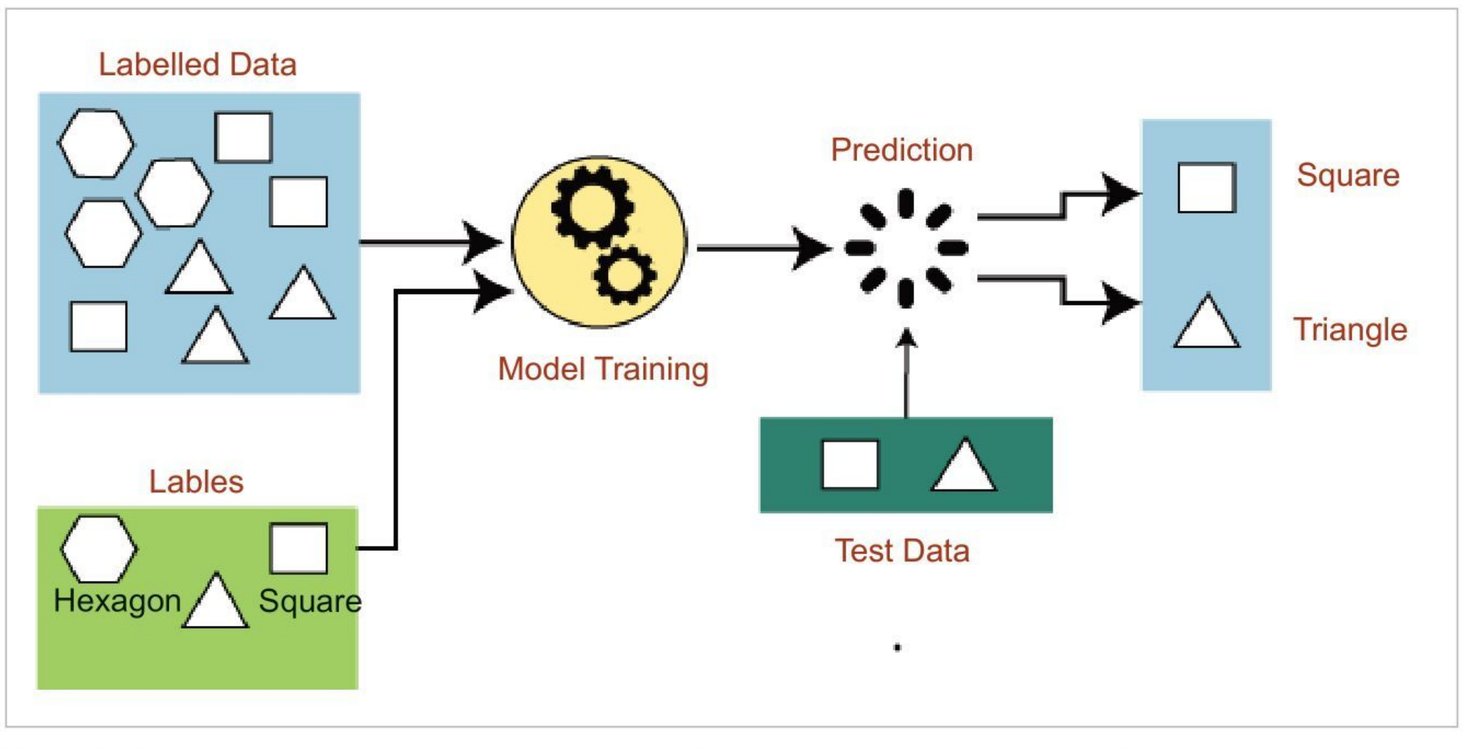


Figure 1: Supervised learning (Credits: *https://www.javatpoint.com/supervised-machine-learning*)

Consider the output variable (y) as the price and the input variable (x) as the age of the wine. Do remember that there could be more than one input variable. To help understand this better, we will run through an example with the values shown in Table 1.

Age (in years)	Price (Rs)
0	75
1.2	100
2.1	220
3.4	300
4.1	440
5.6	500
7	700

Table 1: The age (x) and price (y) of wine

Visualisation

The graphical representation of the data can be seen in Figure 2.

```
# Visualising the Training set
plt.scatter(X, y, color = 'red')
plt.title('Age vs Price of Wine
(Training set)')
plt.xlabel('Price of Wine')
plt.ylabel('Age of Wine')
plt.show()
```

We can easily see that there is a relationship between these two variables. Technically, this suggests that the variables are correlated. The correlation coefficient 'R' determines how strong the correlation is.

Figure 2: Data visualisation

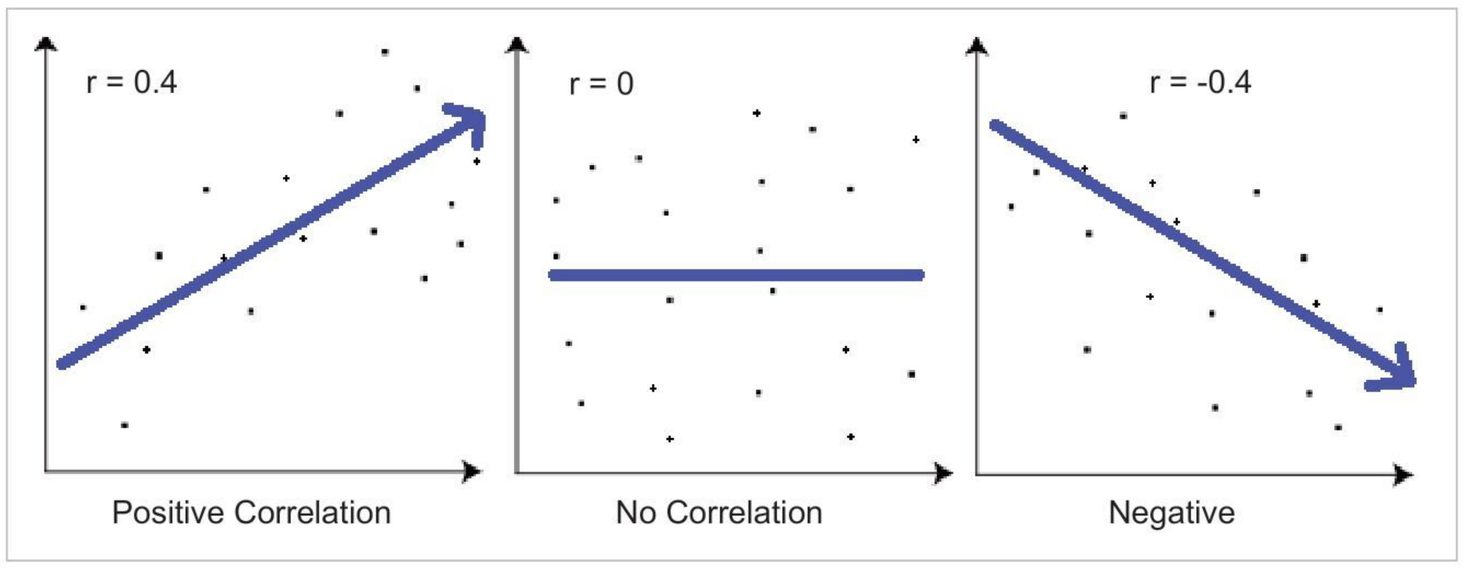


Figure 3: Correlation visualisation (Credits: *https://www.statisticshowto.com/probability-and-statistics/correlation-coefficient-formula/#Pearson)*

R can range from +1 to -1, but the further this figure is away from 0, the stronger is the correlation. '1' indicates a strong positive relationship and '-1' indicates a strong negative relationship. '0', on the other hand, indicates no relationship at all.

A graphical view of a correlation of -1, 0 and +1 is shown in Figure 3.

There are several types of correlation coefficients but the most popular is Pearson's Correlation. Its coefficient formula is seen in Figure 4.

The two variables are divided by the product of the standard deviation of each data sample. This is the normalisation of the covariance between the two variables to give an interpretable score.

Let's switch back to our scenario of age and price of wine to find out whether there is a correlation and, if yes, what is the value of R.

```
from scipy.stats import pearsonr
# calculate Pearson's correlation
corr, _ = pearsonr(list(X.flat), y)
print('Pearsons correlation: %.3f' %
corr)
```

The output seen is:

```
Pearsons correlation: 0.985
```

	Age of wine	Price of wine	$x - \bar{x}$	$y - \bar{y}$	$(x - \bar{x})^2$	$(y - \bar{y})^2$	$(x - \bar{x})*(y - \bar{y})$
1	0	75	-3.34	-258.57	11.15	66859.18	864.37
2	1.2	100	-2.14	-233.57	4.57	54555.61	500.51
3	2.1	220	-1.24	-113.57	1.53	12898.47	141.15
4	3.4	300	0.06	-33.57	0	1127.04	-1.91
5	4.1	440	0.75	106.43	0.56	11327.04	80.58
6	5.6	500	2.26	166.43	5.11	27698.47	375.65
7	7	700	3.66	366.43	13.4	134269.9	1340.08
Sum	23.4	2335	0 (will always be zeros)	0 (will always be zero)	36.36 **is Sxx**	308735.71 **is Syy**	3300.42 **is Sxy**
Average	**3.34 is $\bar{x}$**	**333.57 is $\bar{y}$**					

Table 2

	Age of wine x	Price of wine y	Predicted price of wine $\hat{y} = B0 + B1x$ $\hat{y}$	Residuals (e_ $y - \hat{y}$	Squared residuals $(y - \hat{y})^2$
1	0	75	B0 + B1 * 0	75 - (B0 + B1 * 0)	[75 - (B0 + B1 * 0)] ^ 2
2	1.2	100	B0 + B1 * 1.2	100 - (B0 + B1 * 1.2)	[100 - (B0 + B1 * 1.2)] ^ 2
3	2.1	220	B0 + B1 * 2.1	220 - (B0 + B1 * 2.1)	[220 - (B0 + B1 * 2.1)] ^ 2
4	3.4	300	B0 + B1 * 3.4	300 - (B0 + B1 * 3.4)	[300 - (B0 + B1 * 3.4)] ^ 2
5	4.1	440	B0 + B1 * 4.1	440 - (B0 + B1 * 4.1)	[440 - (B0 + B1 * 4.1)] ^ 2
6	5.6	500	B0 + B1 * 5.6	500 - (B0 + B1 * 5.6)	[500 - (B0 + B1 * 5.6)] ^ 2
7	7	700	B0 + B1 * 7	700 - (B0 + B1 * 7)	[700 - (B0 + B1 * 7)] ^ 2
Sum	23.4	2335	7B0 + 16.4B1	2335 - (7B0 + 23.4B1)	Se
Average	**3.34 is $\bar{x}$**	**333.57 is $\bar{y}$**	**B0 + 3.34B1 is B0 + B1 $\bar{x}$**	**333.57 - (B0 + 3.34B1) is $\bar{y}$ - (B0 + $\bar{x}$B1)**	Se / 7

Table 3

The R value is 0.985, which is close to 1. Hence, the two variables are strongly positively correlated

> ***Note:*** In simple terms, the shape of the data is important to know for selecting the model for prediction. So always draw the plot to get a sense of the data.

Linear regression equation

Now that we have seen that our data is a good use case for linear regression, let's have a look at the formula. The linear equation is:

```
y = B0 + B1*x
```

Here, y is the predicted variable. B0 is the intercept — the predicted value of y when x is 0. In this example, you can see that when x is 0, the value of y is 75. B1 is the regression coefficient, which refers to the change expected in y as x increases. Again going back to the example, as the number of years increases, the price of the wine also increases. Now, let's try and find out how much it increases.

```
y = B0 + B1*x1 + B2*x2 + ... + BN*xN
```

When we have multiple input variables, as seen here, the aim of the linear regression algorithm is to find the best values for B0, B1, .. , BN.

A model 'train, validate and test' set

Before training our model, we need to do a train and validation split.

- *Train:* Set of input and output data for the model to learn.
- *Validate:* Set of input and output data used for evaluation of the model, to see it is not over fitted and reacting well to unseen data. It is also used for fine tuning hyper parameters, feature selection, threshold cut-off selection, and so on.
- *Test:* Final set of unseen data to evaluate the final model.

OLS calculation

In this method, we have to choose the values of B0 and B1 such that the total sum of squares of the difference between the calculated and observed values of y is minimised.

Before we write the source code in Python, we need to understand how the OLS works if we had to do things manually.

$$r = \frac{n(\Sigma xy) - (\Sigma x)(\Sigma y)}{\sqrt{[\, n\Sigma x^2 - (\Sigma x)^2 \,][\, n\Sigma y^2 - (\Sigma y)^2 \,]}}$$

Figure 4: Correlation coefficient formula (Credits: *https://www.statisticshowto.com/probability-and-statistics/correlation-coefficient-formula/#Pearson)*

Step 1: Calculate the data given in Table 2 from the formulas given below, i.e., Sxx – Sum of squares of x; Syy – Sum of squares of y; and Sxy – Sum of products of x and y.

The formulas are:

- x – Mean of x
- y – Mean of y
- Sxx – Sum of squares of (x - $\bar{x}$)
- Syy – Sum of squares of (y - $\bar{y}$)
- Sxy – Sum of products of (x - $\bar{x}$) and (y - $\bar{y}$)

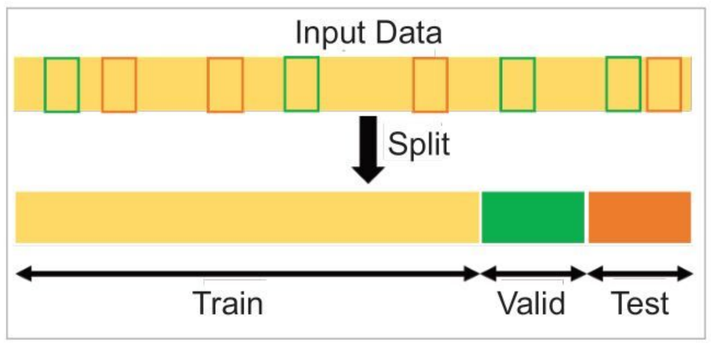


Figure 5: Randomly split the input data into a train, valid, and test set (Credits: *https://towardsdata-science.com/how-to-split-data-into-three-sets-train-validation-and-test-and-why-e50d22d3e54c)*

Table 2 shows the calculation.

Step 2: Find the residual sum of squares, written as Se or RSS.

The formulas are:

- y is the observed value
- ŷ is the estimated value based on our regression equation (*Note:* The caret in ŷ is affectionately called a hat, so we call this parameter estimate y-hat)
- y – ŷ is called the residual and is written as e

The calculation is given in Table 3.

> ***Note:*** How the formula derivation is arrived at is out of the scope of this article. *Hint:* Use calculus to differentiate y = (B0 + B1x)^n with respect to x. Calculus is the mathematical study of continuous change, and differentiation cuts things into smaller pieces to find how they change.

Step 3: Find the regression equation using the B0 and B1 formulas.

- B0 = ¯y - ¯x B1
- B1 = Sxy / Sxx

Let's plug in the values we calculated in Step 1.

- B1 = Sxy / Sxx = 3300.43/36.36 = 90.77
- B0 = ¯y - ¯x B1 = 333.57 – (3.34 * 90.77) = 30.40

The regression equation is:

```
y = 30.40 + 90.77x
```

> ***Note:*** The values above are rounded; hence, if you run through the program there will be slight variations. Also, here we should have normalised the value of the price of wine, to see non-biased results.

Here's a recap of what we have seen so far.

The relationship between the residuals and the slope B1 (alias a) and intercept B0 (alias b) is always as follows.

B1: Sum of products of x and y / sum of squares of x = Sxy / Sxx

- B0: Mean of y – mean of x * B1 = ¯y - ¯x B1

Cost function

This helps us find the best possible values of B0, B1, .., BN, from where the average is taken by all the residuals. It is also known as the mean squared error (MSE) function.

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \tilde{y}_i)^2$$

Figure 6: MSE formula

$$RMSE = \sqrt{\frac{1}{n}\sum_{j=1}^{n}(y_j - \hat{y}_j)^2}$$

Figure 7: RMSE formula

We can also use the root mean squared error (RMSE) function as a cost function.

RMSE is more sensitive to outliers. But when the outliers are exponentially rare (like in a bell shaped curve), the RMSE performs very well and is generally preferred.

Correlation coefficient (R)

Now that we have the regression equation, let's see how accurate our predictions are when we use it. As per our expectation, shown in Figure 2, the line should be closest to the dots, i.e., the predicted values should be closer to the observed values (dots).

To help understand the accuracy of the predictions, we can use the correlation coefficient R to represent the accuracy of a regression equation. This is the same function used before to check the correlation of the data to make sure the linear regression model is a suitable candidate for this data.

The formulas are:

R = sum of products y and ¯y / root of sum of squares of y and sum of squares of ¯y

Data: B1 = Sxy / Sxx = 3300.43/36.36 = 90.77

B0 = ¯y - ¯x B1 = 333.57 - (3.34 * 90.77) = 30.40

The calculation is given in Table 4.

To check how much variance is explained by our regression equation, we use R^2. This is also called the coefficient of determination.

Model training

It's really simple to code in Python, since

	Age of wine x	Price of wine y	Predicted price of wine ŷ = B0 + B1x ŷ = 30.40 + 90.77x	Residuals (e = y - ŷ	Squared residuals (y - ŷ)^2
1	0	75	30.11	44.89	2014.8
2	1.2	100	139.05	-39.05	1524.68
3	2.1	220	220.75	-0.75	0.56
4	3.4	300	338.76	-38.76	1502.24
5	4.1	440	402.3	37.7	1421.04
6	5.6	500	538.47	-38.47	1479.97
7	7	700	665.56	33.44	1186.15
Sum	23.4	2335			
Average	**3.34 is ¯x**	**333.57 is ¯y**			Se / 7 = 9129.42

Table 4

all the functions to be used are already written. We will be using the module named *sklearn* for the task.

```
from sklearn.linear_model import
LinearRegression
lm = LinearRegression()
lm.fit(X_train, y_train)
y_pred = lm.predict(X_test)
```

Here, *X_train* is the training input; *y_train* is training expected output; *X_test* is testing input; *y_test* is testing actual output; and *y_pred* is testing predicted output.

The methods used are:

- *LinearRegression():* This fits a linear model with coefficients B = (B1, …, BN) to minimise the residual sum of squares between the observed targets in the data set and the targets predicted by the linear approximation.
- *fit():* This method fits the model to the input training instances.
- *predict():* This performs predictions on the testing instances, based on the parameters learned during `fit`. The output source code is:

```
• y_pred
  • array([ 30.11355599,
    139.04715128, 220.74734774,
    338.75874263,402.30333988,
    538.47033399, 665.55952849])

• print("Coeffient(B1) {} and
  Intercept(B0) {}".format(lm.coef_,
  lm.intercept_))
  • Coeffient(B1) [90.77799607]
    and Intercept(B0)
    30.11355599214147
```

Predictions

Now that we have trained our model, let's see how close to the truth the predictions are.

```
plt.title('Prediction of Price of Wine
based on the given Age')
plt.xlabel("Age of wine") #x label
plt.ylabel("Price of wine") #y label
```

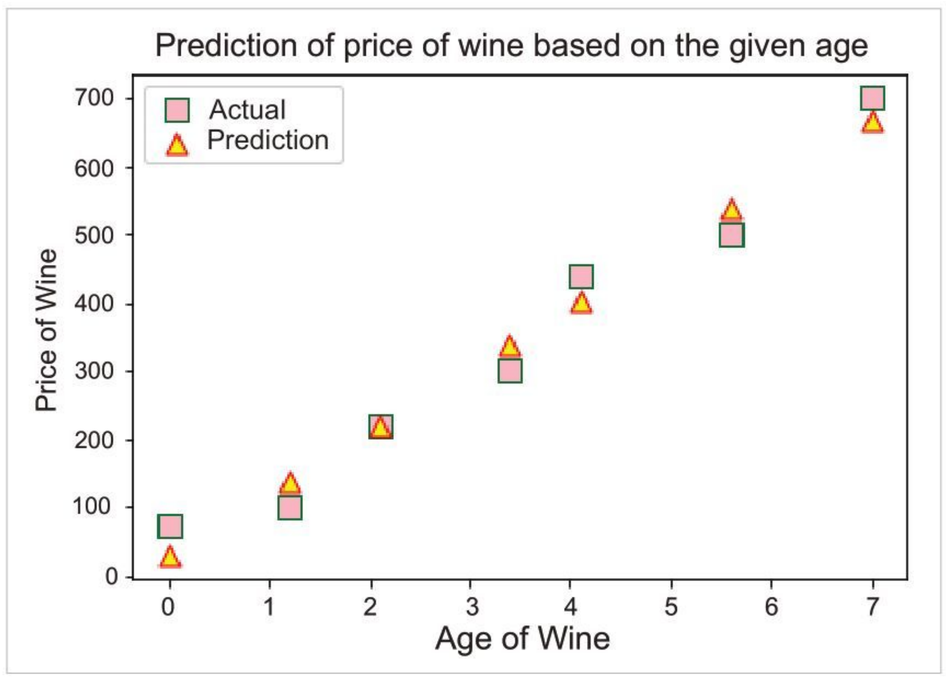


Figure 8: Prediction vs actual relationship

```
plt.scatter(X_test, y_test, c ="pink",
            marker ="s",
            edgecolor ="green",
            s = 100,
            label = "Actual")
plt.scatter(X_test, y_pred, c
="yellow",
            marker ="^",
            edgecolor ="red",
            s = 80,
            label = "Prediction")
plt.legend()
```

Though the model performed really well, the predictions were not completely accurate. So how confident are we about our model's prediction? To answer this question, we need to know the 'confidence interval'.

Generally, we don't want our model to be 100 per cent accurate, because it will be considered as an overfit and will not generalise well.

In simple terms, assume the model is a student, who is going for an English language exam. We have given this student a few questions to learn which, coincidently, are asked in the exam. The model does well, scoring full marks.

However, this will not always be the case. If the questions are different, the model will not understand them and fail, as it has memorised only these answers verbatim.

The takeaway from here is that instead of giving a few questions, giving the whole book (i.e., more data) to the model will make memorising hard, and it will start learning and finding patterns.

The assumptions we make here are:

- Input and output variables should be numeric.
- Outliers should be removed or replaced.
- Remove collinearity between input variables (x), as linear regression will over-fit your data when you have highly correlated input variables. There should be collinearity with input and output but not within input variables.

Rescale inputs, as linear regression will often make more reliable predictions if you rescale input variables using standardisation or normalisation. END

References

[1] *https://towardsdatascience.com/introduction-to-machine-learning-algorithms-linear-regression-14c4e325882a*
[2] *https://machinelearningmastery.com/linear-regression-for-machine-learning/*
[3] *https://scikit-learn.org/stable/auto_examples/linear_model/plot_ols.html#sphx-glr-auto-examples-linear-model-plot-ols-py*
[4] *https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html*
[5] The Manga Guide to Regression Analysis

By: Thangaselvi Arichandrapandian

The author works in a leading bank as AVP. She is a perennial learner and loves the quote: "The more I learn, the more I realise how much I don't know."
- *Albert Einstein.*

User Interfaces in R

We have covered a lot of material in the R programming language syntax such as data structures, and import and transformation of data. In this sixth article in the 'R, Statistics and Machine Learning' series, we shall explore the various user interfaces available to interact with R.

We will be using R version 4.1.0 installed on Parabola GNU/Linux-libre (x86-64) for the code snippets.

```
$ R --version
R version 4.1.0 (2021-05-18) -- "Camp Pontanezen"
Copyright (C) 2021 The R Foundation for Statistical Computing
Platform: x86_64-pc-linux-gnu (64-bit)

R is free software and comes with absolutely no warranty.
You are welcome to redistribute it under the terms of the GNU
General Public License versions 2 or 3. For more information
on these matters, see https://www.gnu.org/licenses/.
```

R console

We have already used the R console environment to learn the semantics of the R programming language. The console provides a REPL (Read-Eval-Print Loop) to iteratively develop software, and is very useful when experimenting or testing code. You can start an R console session using the *R* command, and it will provide a prompt. You can exit the session using the *quit()* function as shown below:

```
$ R
R version 4.1.0 (2021-05-18) -- "Camp Pontanezen"
Copyright (C) 2021 The R Foundation for Statistical Computing
Platform: x86_64-pc-linux-gnu (64-bit)
R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.
   Natural language support but running in an English locale
R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.
 Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.
> quit()
Save workspace image? [y/n/c]: n
```

The command line R execution supports multiple arguments, as can be seen from the *help* option:

```
$ R --help
```

The output of the command will be displayed on your screen with all the details.

Batch mode

R provides a batch mode, where you can script a series of commands from a source file. For example, we can write a script to read the *bank.csv* file from the 'Bank Marketing Data Set' available from the UCI machine learning repository available at *https://archive.ics.uci.edu/ml/datasets/Bank+Marketing*. You can display the first few rows and print a summary of the data in the following *batch.R* file:

```
bank <- read.csv(file="bank.csv", sep=";")
head(bank)
print(summary(bank))
```

You can execute the script using the following command from a bash shell prompt:

```
$ R CMD BATCH batch.R
```

By default, the results are stored in a *.Rout* file. The time

taken to run the script is also recorded in the output.

```
$ cat batch.Rout
R version 4.1.0 (2021-05-18) -- "Camp Pontanezen"
Copyright (C) 2021 The R Foundation for Statistical Computing
Platform: x86_64-pc-linux-gnu (64-bit)
R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.
  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.
Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.

Type 'q()' to quit R.

[Previously saved workspace restored]
> bank <- read.csv(file="bank.csv", sep=";")
>
> head(bank)
  age         job marital education default balance housing
loan  contact day
1 30  unemployed married  primary  no 1787 no  no cellular 19
2  33 services married secondary no 4789 yes yes cellular  11
3  35  management single tertiary no 1350 yes no cellular  16
4  30  management married tertiary no 1476 yes yes unknown  3
5  59 blue-collar married secondary no 0  yes no  unknown   5
6  35  management  single  tertiary no 747 no no cellular  23

  month duration campaign pdays previous poutcome  y
1   oct       79        1    -1        0  unknown no
2   may      220        1   339        4  failure no
3   apr      185        1   330        1  failure no
4   jun      199        4    -1        0  unknown no
5   may      226        1    -1        0  unknown no
6   feb      141        2   176        3  failure no
>
> print(summary(bank))
      age            job              marital
education
 Min.   :19.00  Length:4521     Length:4521       Length:4521
 1st Qu.:33.00 Class :character Class :character Class
:character
 Median :39.00   Mode  :character   Mode  :character   Mode
:character
 Mean   :41.17
 3rd Qu.:49.00
 Max.   :87.00
   default             balance        housing
loan
 Length:4521        Min.   :-3313   Length:4521
Length:4521
 Class :character   1st Qu.:   69   Class :character   Class
:character
 Mode  :character   Median :  444   Mode  :character   Mode
:character
                    Mean   : 1423
                    3rd Qu.: 1480
                    Max.   :71188
  contact         day           month              duration
 Length:4521     Min.  : 1.00   Length:4521        Min.   :   4
 Class :character  1st Qu.: 9.00  Class :character 1st Qu.: 104
 Mode  :character Median :16.00   Mode :character  Median : 185
                  Mean   :15.92                    Mean   : 264
                  3rd Qu.:21.00                    3rd Qu.: 329
                  Max.   :31.00                       Max.   :3025
    campaign        pdays        previous        poutcome
 Min.   : 1.000   Min.   : -1.00   Min.   : 0.0000
Length:4521
 1st Qu.: 1.000   1st Qu.: -1.00   1st Qu.: 0.0000   Class
:character
 Median : 2.000   Median : -1.00   Median : 0.0000   Mode
:character
 Mean   : 2.794   Mean   : 39.77   Mean   : 0.5426
 3rd Qu.: 3.000   3rd Qu.: -1.00   3rd Qu.: 0.0000
 Max.   :50.000   Max.   :871.00   Max.   :25.0000
      y
 Length:4521
 Class :character
 Mode  :character

>
> proc.time()
   user  system elapsed
  0.160   0.036   0.144
```

You can also use the *Rscript* command to execute the above script, as shown below:

```
$ Rscript batch.R
```

The R commands can also be written to a source file with the shebang ("#!") operator. For example:

```
#! /usr/bin/env Rscript
print("From executable Rscript!");
```

The script can be executed as follows:

```
$  ./batch_print.R
[1] "From executable Rscript!"
```

GNU Emacs

The Emacs Speaks Statistics (ESS) package provides support to work with statistical packages. It also has support to work with R code and GNU Emacs. You can add the following to your Cask file, and install the ESS package:

```
(depends-on "ess")
```

You can add the following to your GNU Emacs initialisation file in order to use the R mode support:

```
(require 'ess-r-mode)
```

The *M-x ess-version* command provides the installed version number, which is 20211204.958 from the ELPA repository on GNU Emacs 27.2. You can start an R session from Emacs using *M-x R*. Emacs will prompt for a starting working directory, and a new buffer with an R prompt will be displayed, as shown below:

```
R version 4.1.0 (2021-05-18) -- "Camp Pontanezen"
Copyright (C) 2021 The R Foundation for Statistical Computing
Platform: x86_64-pc-linux-gnu (64-bit)
R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale
R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.
Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.
> setwd('/home/guest/working/')
>
```

A number of commands have been implemented to interact between R code and the ESS process. A summary of the available commands to evaluate code is given in Table 1.

The advantage of using an R session within GNU Emacs is that you can save the session for future reference. You can also use ESS with the TRAMP (transparent remote access, multiple protocols) package to start a remote session. The same can be initiated with the *ess-remote* command. Do read the 'ESS - Emacs Speaks Statistics' PDF manual to learn more about its use.

The Babel project provides the ability to execute R code in an Emacs org file. The following code snippet enables support for R Babel and needs to be included in the Emacs initialisation file:

Command	Comments
ess-eval-region-or-line-and-step	Evaluate highlighted region or current line and step to next line of code
ess-eval-region-or-function-or-paragragh	Send selected region or function or paragraph
ess-eval-line	Send current line
ess-eval-line-and-go	Send line and switch point to the ESS process
ess-eval-function	Send function
ess-eval-function-and-go	Send function and switch point to the ESS process
ess-eval-buffer	Send the current buffer
ess-eval-buffer-and-go	Send current buffer and switch point to the ESS process

Table 1

```
(org-babel-do-load-languages
 'org-babel-load-languages
 '((R . t)))
```

You can enclose R code within the org Babel code blocks and execute the same in an Emacs org file to obtain the results as shown below:

```
#+BEGIN_SRC R
  1 < 2
  #+END_SRC

  #+RESULTS:
  : TRUE
```

RStudio

RStudio is an integrated development environment (IDE) for the R programming language. It can be used as a desktop application, as well as a server that can be accessed on a browser. It is primarily written in Java, C++ and JavaScript, and the open source edition is available at *https://github.com/rstudio/rstudio.* Refer to Figure 1 for RStudio.

Shiny

Shiny is an R package that is useful to develop interactive Web applications with R. You can install it with your R environment using:

```
> install.packages("shiny")
  ...
  * installing *source* package 'shiny' ...
  ** package 'shiny' successfully unpacked and MD5 sums
checked
  ** using staged installation
```

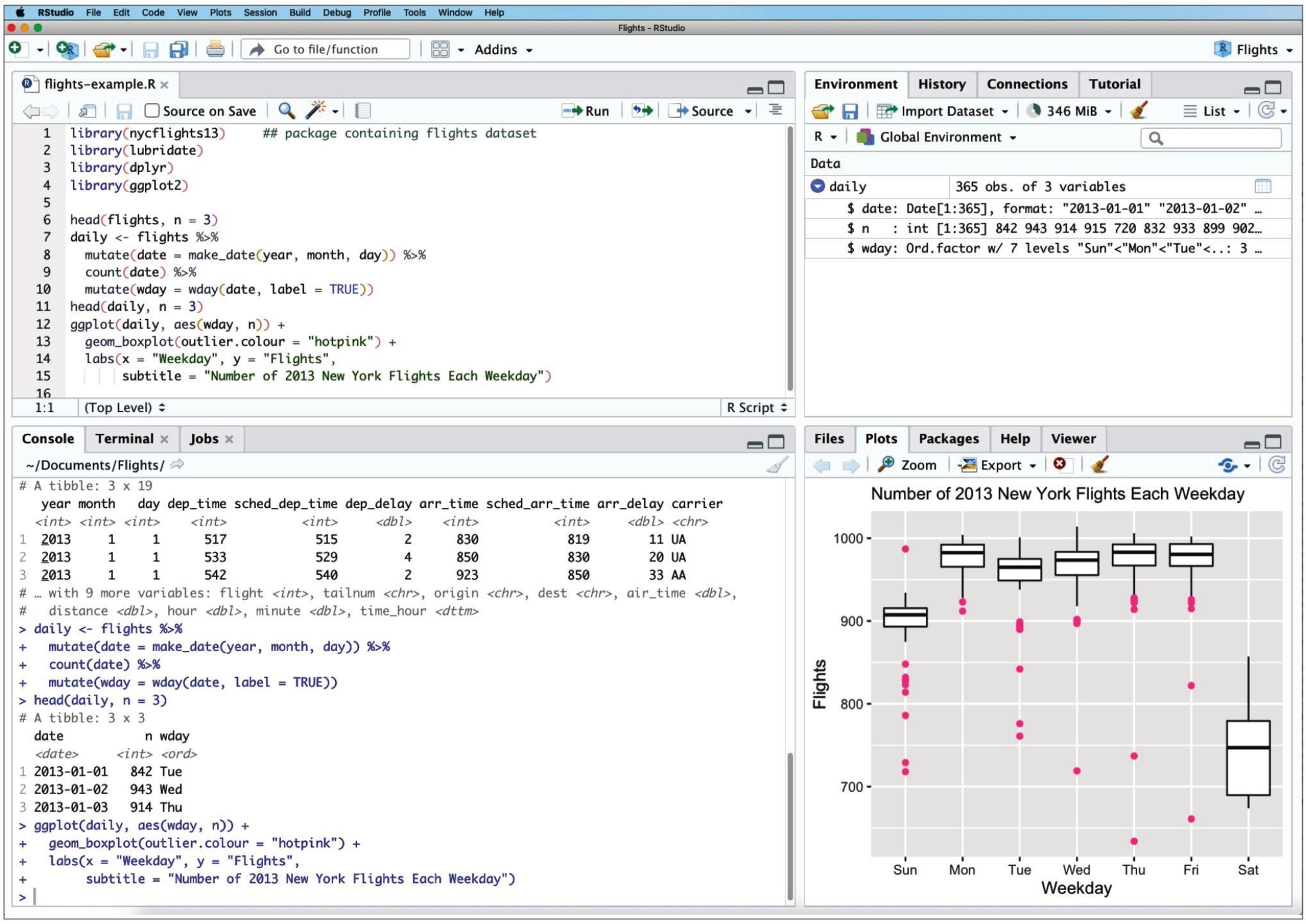


Figure 1: RStudio

```
  ** R
  ** inst
  ** byte-compile and prepare package for lazy loading
  ** help
  *** installing help indices
  *** copying figures
  ** building package indices
  ** testing if installed package can be loaded from
temporary location
  ** testing if installed package can be loaded from final
location
  ** testing if installed package keeps a record of temporary
installation path
  * DONE (shiny)
```

There are pre-built examples and demos provided by the installed Shiny package. You can test one of the examples using the following commands:

```
> library(shiny)
> runExample("02_text")
 Listening on http://127.0.0.1:3535
```

The app listens on a port and you need to set the default R browser in */etc/R/Renviron*. For example, on Parabola GNU/Linux-libre, the setting is updated to use the Iceweasel (Firefox) browser as follows:

```
## Default browser
  R_BROWSER=${R_BROWSER-'/usr/bin/iceweasel'}
```

The source code for the app defines the user interface and the server logic. A fluid layout is used for the panel, and the input selection consists of drop-down menus and text boxes. The server logic uses a reactive programming style to handle the input and displays the data.

```
library(shiny)
# Define UI for dataset viewer app ----
ui <- fluidPage(
  # App title ----
  titlePanel("Shiny Text"),
  # Sidebar layout with a input and output definitions ----
  sidebarLayout(
    # Sidebar panel for inputs ----
    sidebarPanel(
```

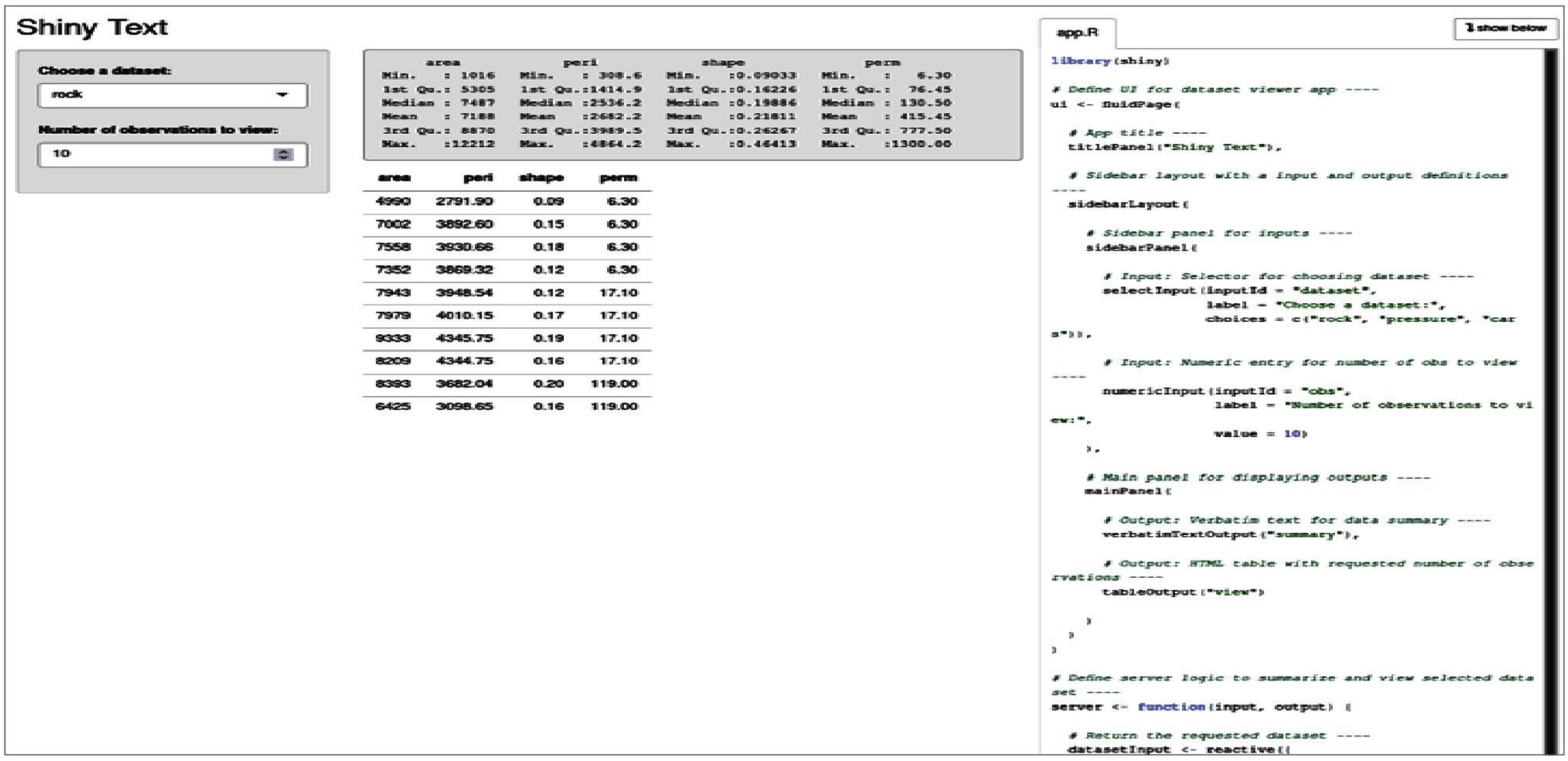


Figure 2: Web screenshot of the text app

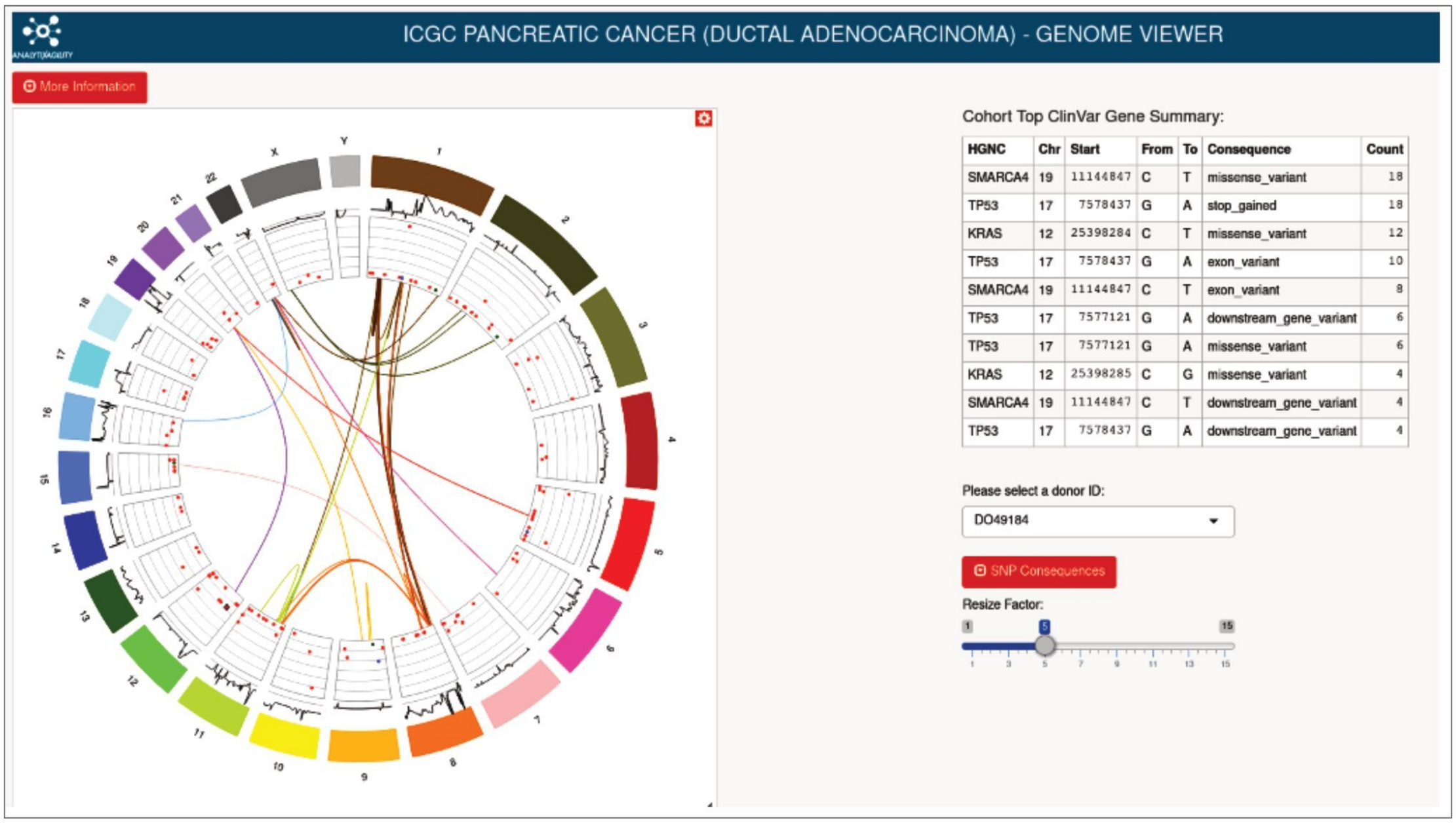


Figure 3: Apps created by Shiny

```
# Input: Selector for choosing dataset ----
selectInput(inputId = “dataset”,
            label = “Choose a dataset:”,
            choices = c(“rock”, “pressure”, “cars”)),
# Input: Numeric entry for number of obs to view ----
numericInput(inputId = “obs”,
                label = “Number of observations to view:”,
                value = 10)
),
# Main panel for displaying outputs ----
mainPanel(
  # Output: Verbatim text for data summary ----
```

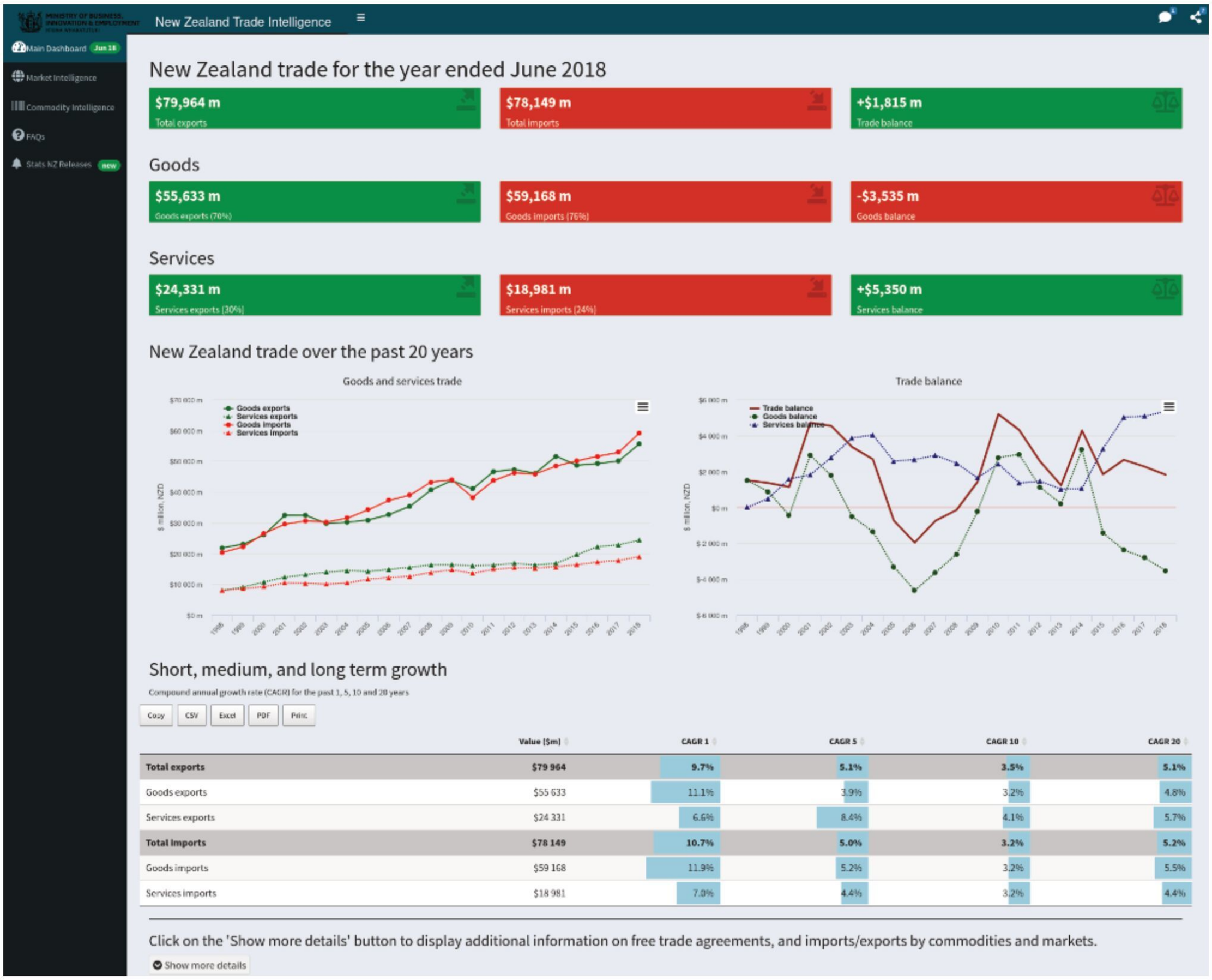


Figure 4: Trading app in Shiny

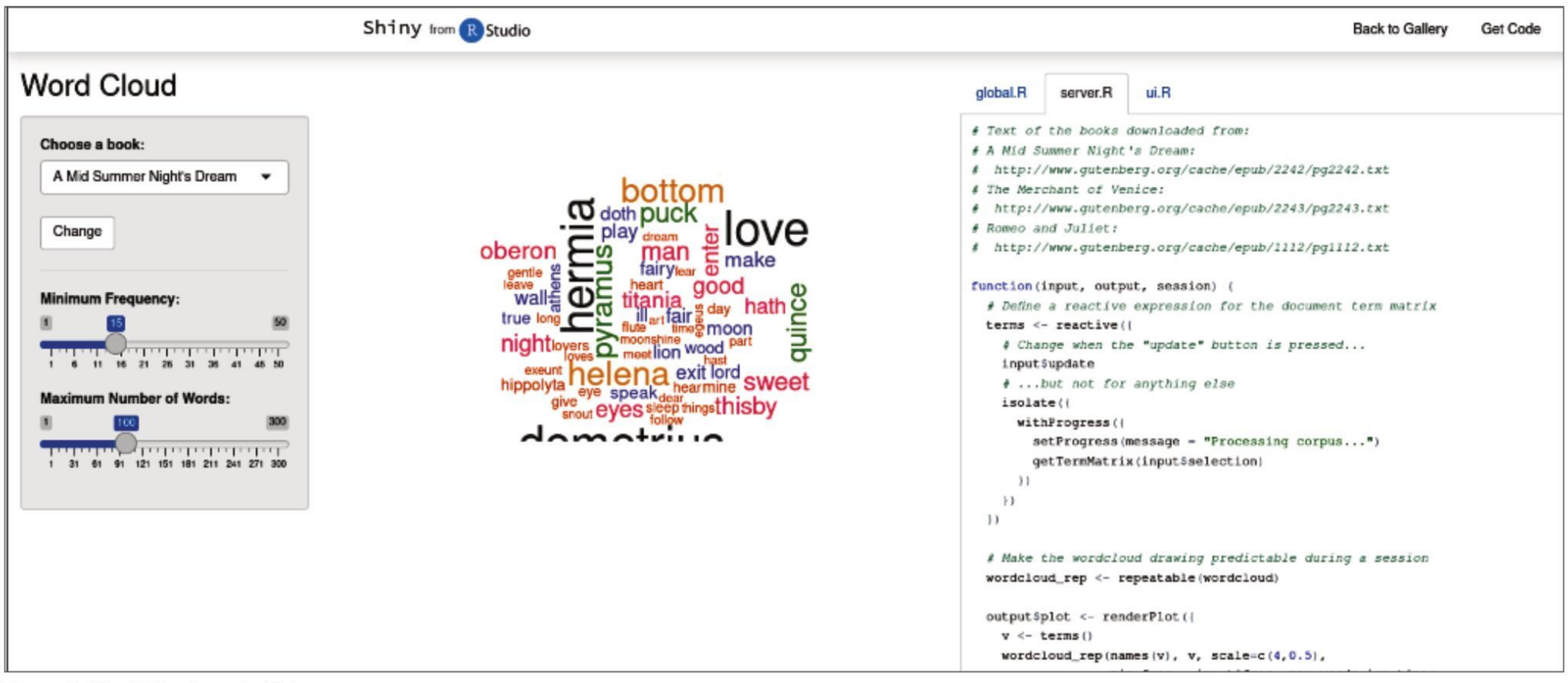


Figure 5: Word Cloud app in Shiny

```
      verbatimTextOutput("summary"),
      # Output: HTML table with requested number of
observations ----
      tableOutput("view")
    )
  )
)
# Define server logic to summarize and view selected dataset
----
server <- function(input, output) {
  # Return the requested dataset ----
  datasetInput <- reactive({
    switch(input$dataset,
           "rock" = rock,
           "pressure" = pressure,
           "cars" = cars)
  })
  # Generate a summary of the dataset ----
  output$summary <- renderPrint({
    dataset <- datasetInput()
    summary(dataset)
  })
  # Show the first "n" observations ----
  output$view <- renderTable({
    head(datasetInput(), n = input$obs)
  })
}
# Create Shiny app ----
shinyApp(ui = ui, server = server)
```

It is recommended to go through the Shiny tutorial videos and the code examples to learn how to best use Shiny for your data sets. END

References

[1] [Moro et al., 2014] S. Moro, P. Cortez and P. Rita, A Data-Driven Approach to Predict the Success of Bank Telemarketing. Decision Support Systems, Elsevier, 62:22-31, June 2014

[2] Emacs Speaks Statistics, *https://ess.r-project.org/*

[3] ESS - Emacs Speaks Statistics Manual, *https://ess.r-project.org/ess.pdf*

[4] RStudio, *https://www.rstudio.com/*

[5] Shiny Tutorial, *https://shiny.rstudio.com/tutorial/*

[6] Shiny Gallery, *https://shiny.rstudio.com/gallery/*

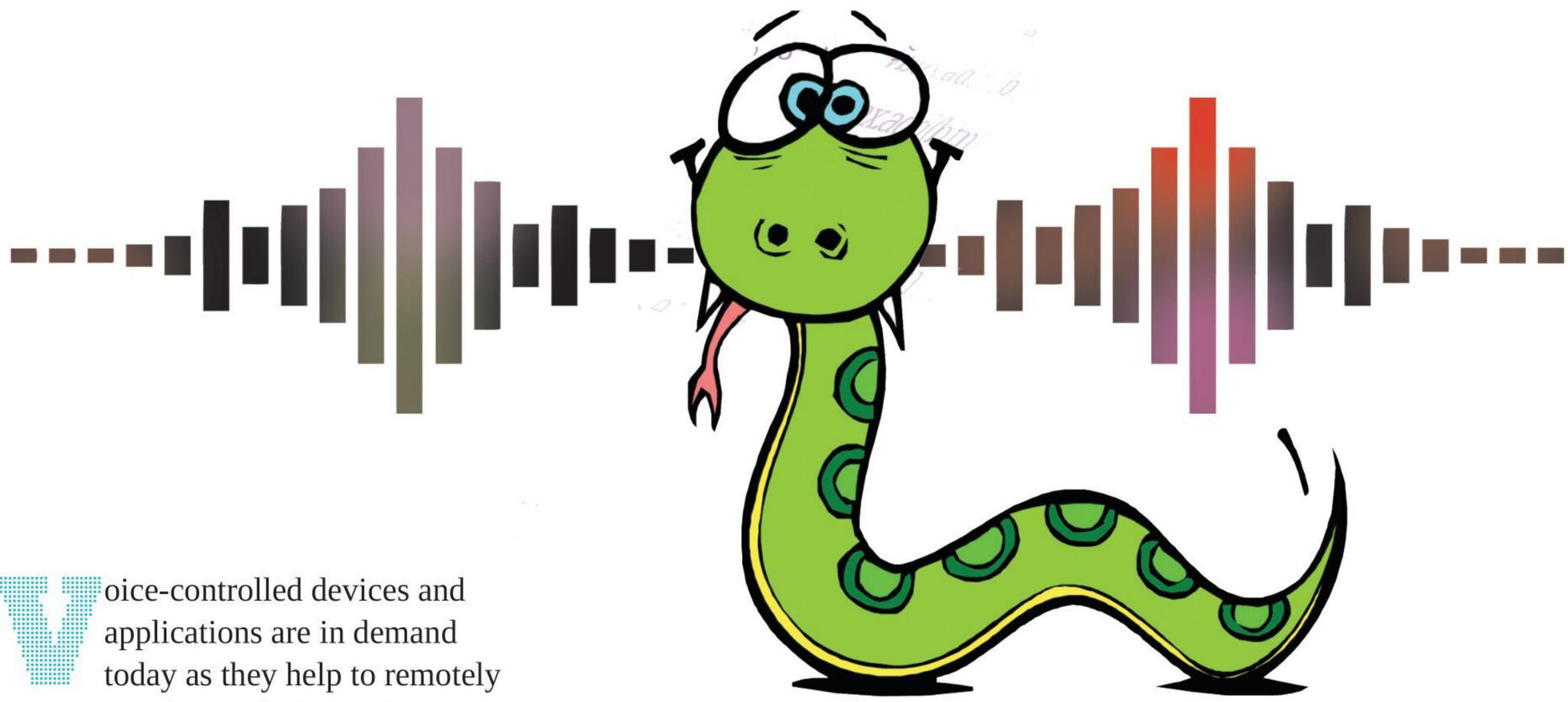
By: Shakthi Kannan

The author is a free software enthusiast.

OSFY Magazine Attractions During 2021-22	
Month	**Theme**
March 2021	AI, Deep learning and Machine Learning
April 2021	DevOps Special
June 2021	Cyberscurity, Open Source Firewall and Network security and Monitoring
July 2021	Mobile/Web App Development and Optimisation
August 2021	Blockchain and Open Source
September 2021	Data Management, Storage and Backup
October 2021	Cloud Special: BigData, Hadoop, PaaS, SaaS, Iaas and Cloud, Automation in Cloud
November 2021	Open Source on Windows
December 2021	Database management and Optimisation
January 2022	Open Source and IoT and Edge
February 2022	Alternatives: Best in the world of Open Source (Tools and Services)

How Python Can Help to Process Audio Waves

Python is one of the most versatile and capable computer programming languages in use today. This article dives into the world of audio waves and plays with them using Python.



Voice-controlled devices and applications are in demand today as they help to remotely access smart home devices, give navigation commands to robots and autonomous vehicles, and so on. Audio signal processing is booming, be it for industrial applications, health care, the financial sector, etc. In this article, we are going to get an insight into the use of Python tools for audio processing. We begin with an overview of this field and then move to the implementation aspects.

An audio signal can be categorised into: (i) speech uttered by humans (also referred to as voice), (ii) music generated by an instrument, (iii) other sounds such as a dog barking, birds chirping, surrounding noise including environmental noise, machinery sounds, etc (Figure 1). In this article, we restrict ourselves to speech and music signals; however, the processing methodology of all such files remains the same.

Speech perception in humans

Speech perception refers to the hearing ability of humans, and is mainly done through the ears. However, the structure of the human ear is complex. The outer ear is what we see. The middle ear is like a path for sound to travel towards the inner ear. The inner ear contains the most sensitive part of the human perception system, known as the cochlea. It is a snail-like structure that contains basilar membranes which act like frequency receptors. The hair cells in these membranes activate at specific frequencies, and are arranged in a logarithmic scale such that they have higher resolution at lower frequencies and lower resolution at higher frequencies. Due to this arrangement, humans find it difficult to distinguish between two different frequency tones in the higher ranges. For example, it is easy to perceive the tonal difference between two sounds having frequencies 100Hz and 200Hz, which are 100Hz apart. However, it becomes almost impossible for humans to distinguish between two sounds with frequencies 10kHz and

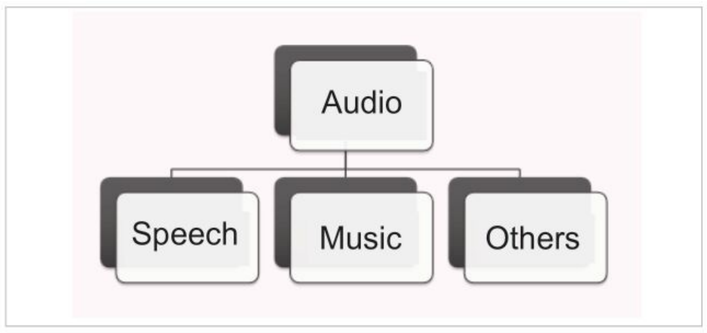


Figure 1: Classification of audio signals

10.1kHz, which are also 100Hz apart. This phenomenon of human auditory nerves is imitated by Mel frequency cepstral coefficients (MFCC) features. Human ears can perceive sounds that range in frequency from 20Hz to 20kHz.

Concept of framing

An audio signal is a digital signal which comprises multiple instantaneous frequencies. Thus, it is a non-stationary signal. In order to apply signal processing algorithms, we need to divide the entire signal into short duration segments, referred to as frames. These frames are typically of 20-25ms duration with an overlap of 10ms. For each frame, a stationarity assumption is made so that the feature extraction algorithms may be executed on them. A segment larger than 30ms may not preserve the stationarity principle due to the involvement of multiple frequency components, and segment length lower than 20ms may not be adequate enough to extract features properly. Researchers experiment on frame durations ranging from 20ms to 30ms, and choose the value that optimises their solution. The framing process of an audio signal is shown in Figure 2.

The number of frames for an audio signal is determined by the duration of the speech.

```
Total # frames = Total duration of the
signal (in sec) / frame size (ms)
```

For example, if the frame duration is 25ms and the signal is 3 seconds long, then this signal will be represented by 120 frames.

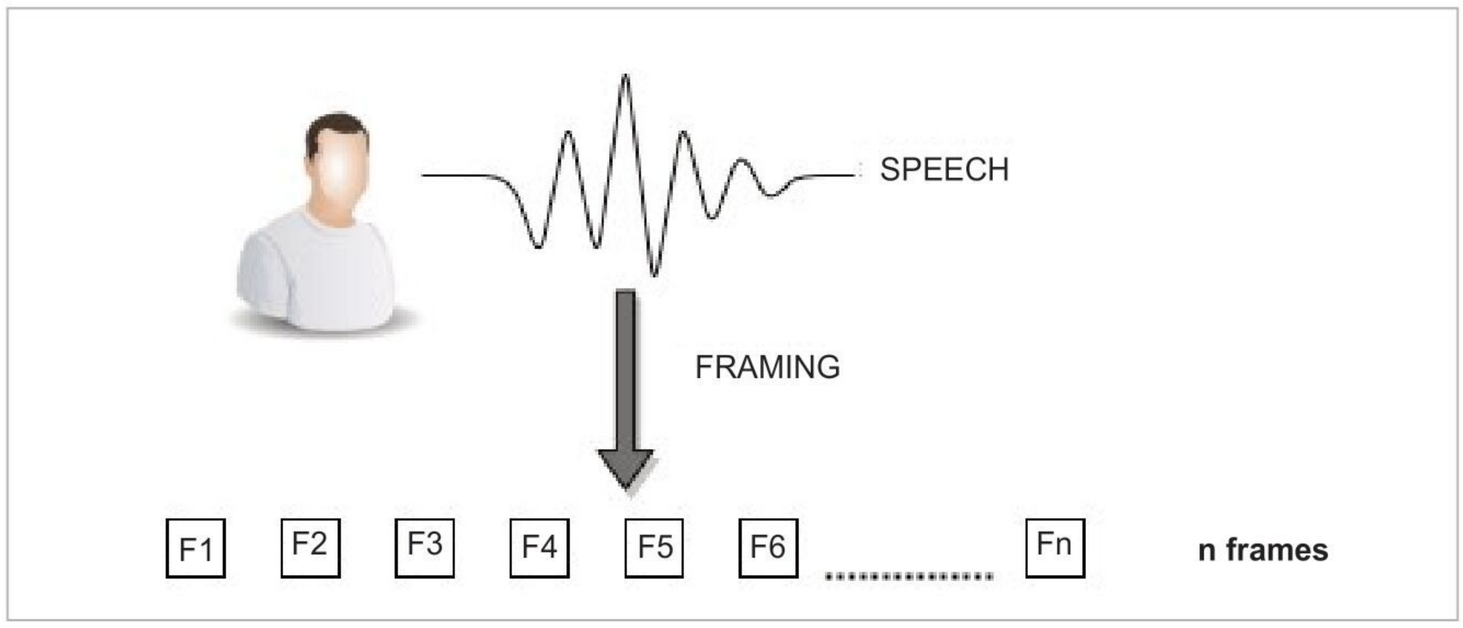


Figure 2: Illustration of framing process in audio signals [10]

Librosa

Librosa is a Python based toolkit that provides several utilities to handle audio files. It can be used to load the audio and plot the waveform of the audio file, by plotting spectrograms and varying the audio files. Let's now see how the visualisation of a sound file is done using Librosa.

To plot the waveform of an audio file, we first need to load the audio and then pass it to the plot *waveplot* function. Waveplot tells us the amplitude of sound around various time intervals. In the following code, the file name can be replaced with the actual name of the *wav* file.

```
Import librosa
file=librosa.load('filename')
librosa.display.waveplot(file)
```

Amplitude envelope

The amplitude envelope extracts the maximum amplitude within each frame and merges these together. It is important to remember that the amplitude represents the loudness of the signal. First, we split the signal into its constituent frames and find the maximum amplitude within each frame. From there, we plot the maximum amplitude in each frame along with time. The amplitude envelope can be used for onset detection or detecting the beginning of the sound. This could be external noise, someone speaking or the beginning of a musical note. The main drawback of the amplitude envelope is that it is sensitive to the outliers.

```
def amplitude_envelope(signal,frame_
length,hop_length):
     max_amplitude_frame=[]
     for i in range(0,len(signal),hop_
length):
         max_amplitude_frame.
append(max(signal[i:i+frame_length]))
  return np.array(max_amplitude_frame)
```

Root mean square energy (RMS)

This is somewhat similar to the amplitude envelope. As opposed to detecting the beginning of a sound, it aims at perceiving the loudness for

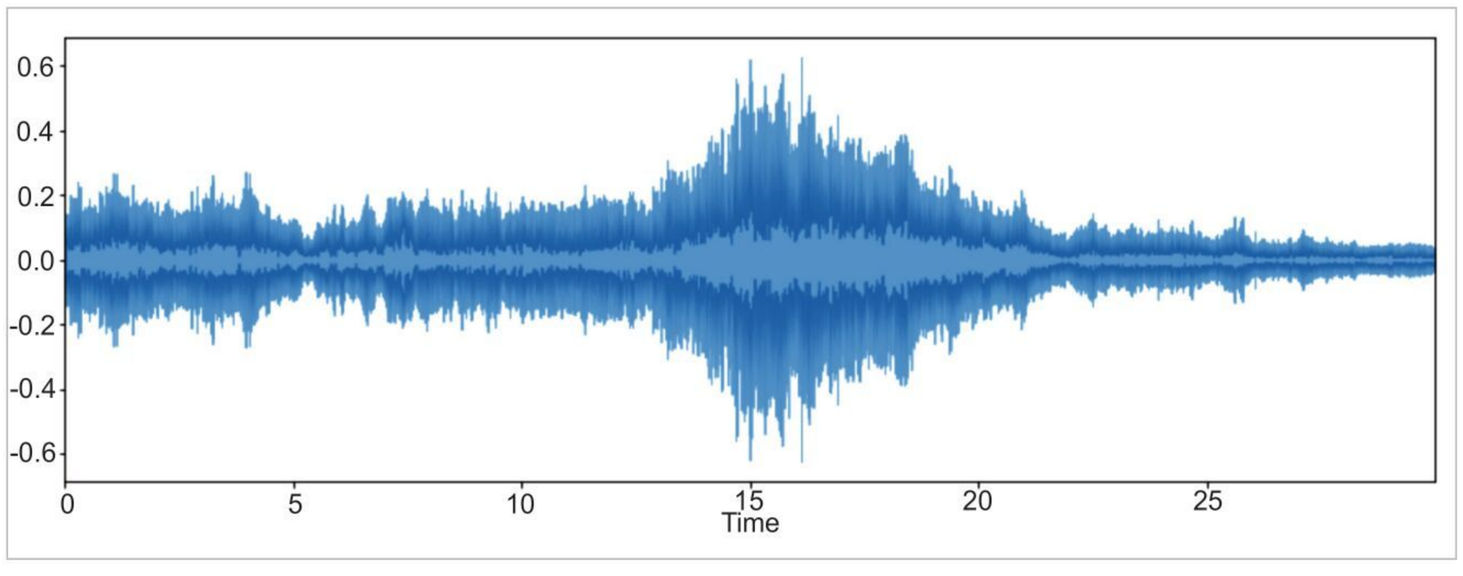


Figure 3: Time domain representation of the audio file

event detection. Also, it is not sensitive to outliers like amplitude envelopes. To calculate RMS energy within a frame, we first square the amplitudes within the frame and then sum them up. We then divide this figure by the total number of samples in the frame and take the square root, which gives us the RMS energy of that specific frame.

```
librosa.feature,rms(signal,frame_
length,hop_length)
```

Zero crossing rate (ZCR)

The ZCR studies the rate at which a signal's amplitude changes within a frame. It can be used for identifying the percussion sound in music, as signals of that type of sound often have a fluctuating crossing rate. However, this feature is often used in human speech detection. To calculate the ZCR in a specific frame, we apply the sigmoid function to two continuous samples and divide their summation by 2.

```
librosa.feature,zero_crossing_
rate(signal,frame_length, hop_length)
```

Fourier transform and spectrogram

The features described above do not provide any information on the frequency of the given signal. To get that information, we apply the discrete Fourier transform on the audio signal, which provides us with the frequency and magnitude of each frequency component in the audio signal. The code for plotting the magnitude spectrum of our audio signal is given below:

```
def plot_magnitude_spectrum(signal, sr,
title, f_ratio=1):
    X = np.fft.fft(signal)
    X_mag = np.absolute(X)
    plt.figure(figsize=(18, 5))
    f = np.linspace(0, sr, len(X_mag))
    f_bins = int(len(X_mag)*f_ratio)
    plt.plot(f[:f_bins], X_mag[:f_bins])
    plt.xlabel('Frequency (Hz)')
    plt.title(title)
```

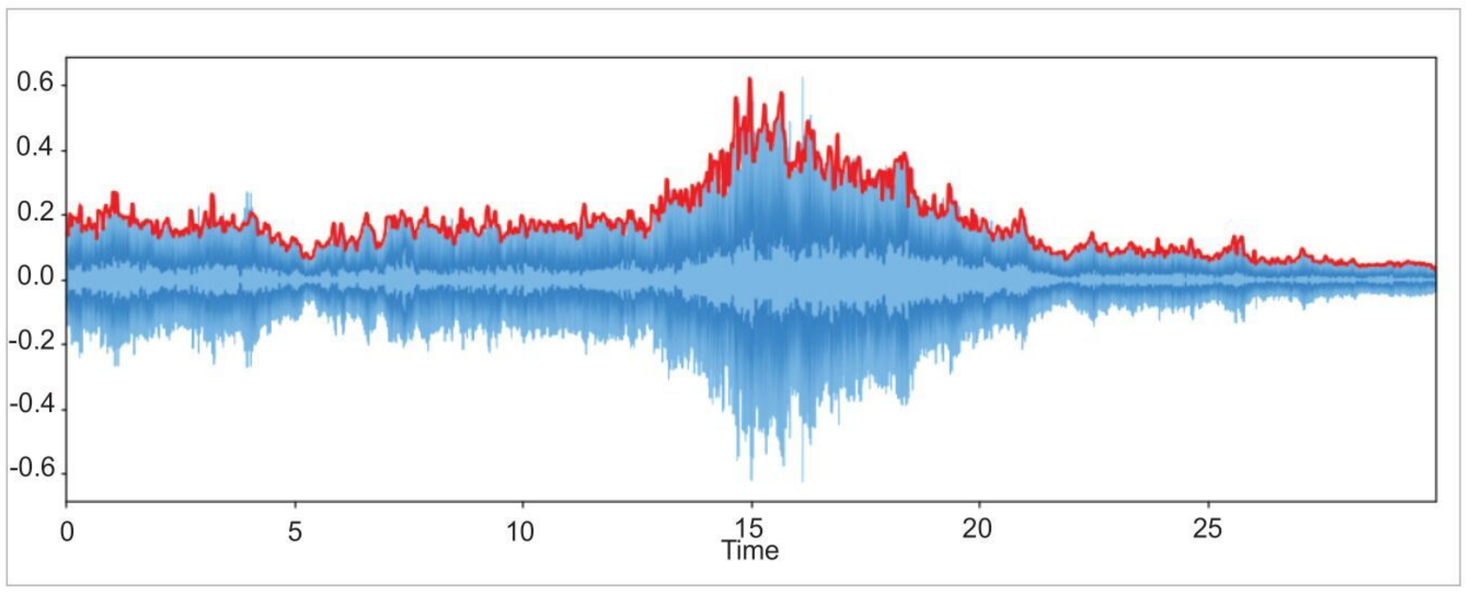


Figure 4: Amplitude envelope of the *wav* file

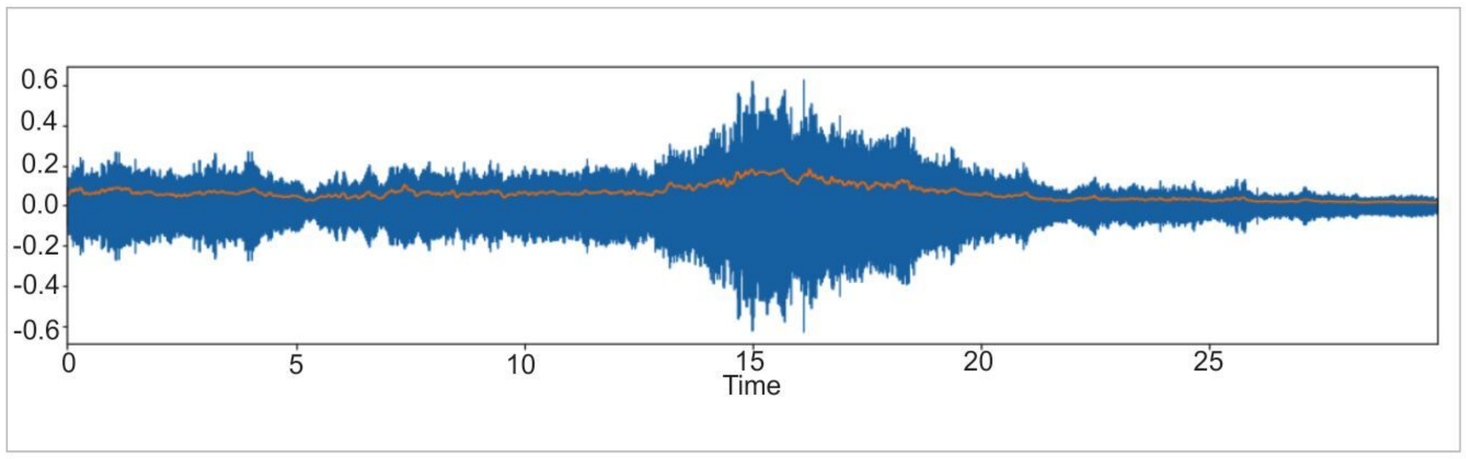


Figure 5: RMS energies of the signal

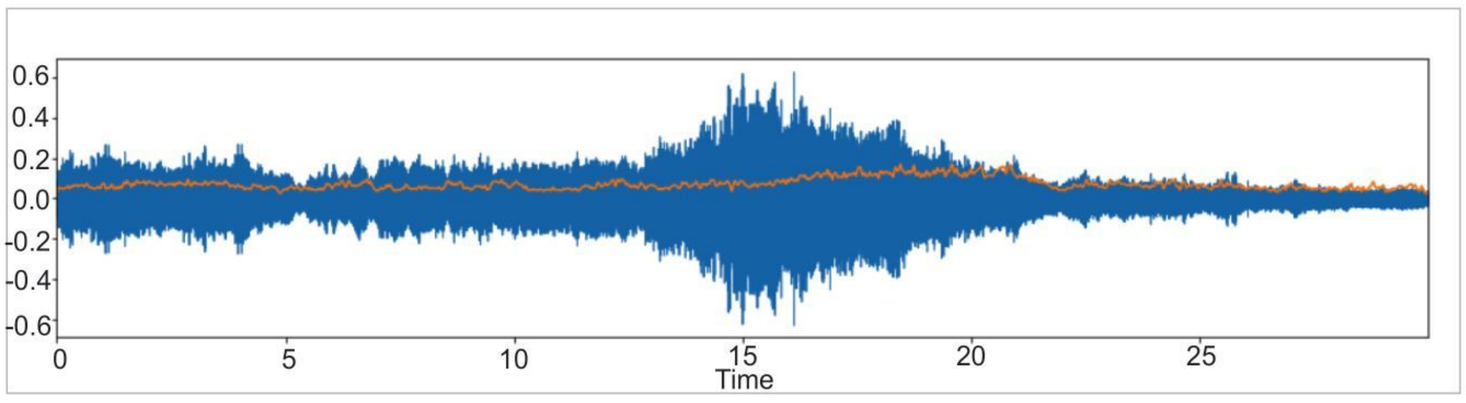


Figure 6: ZCR of an audio signal

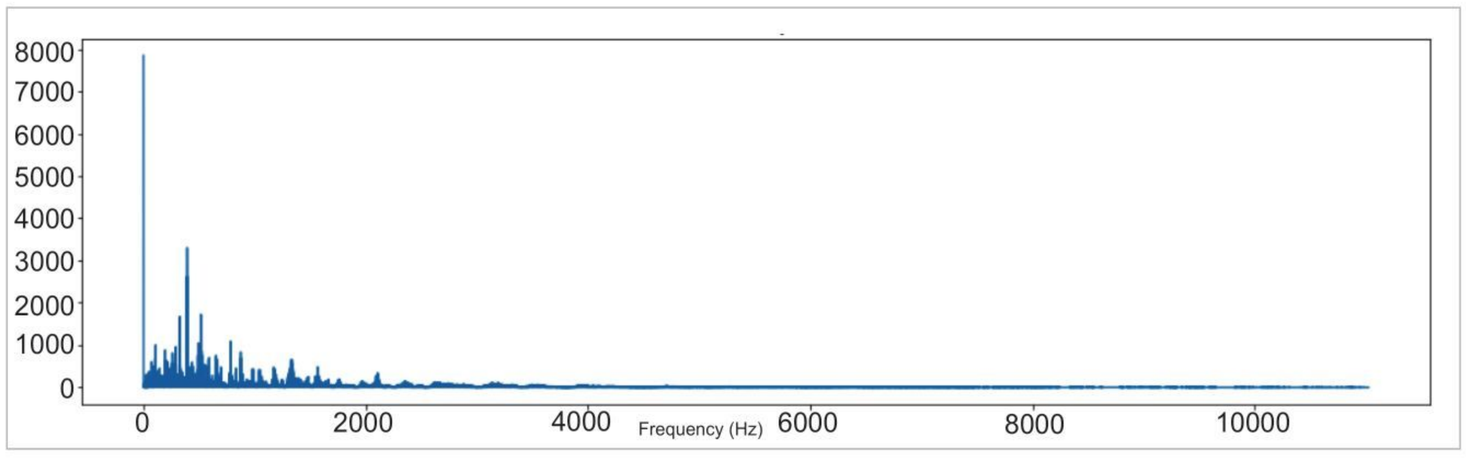


Figure 7: Magnitude spectrum of a signal

Spectrograms using Librosa

The frequency spectrum (Figure 7) shows the frequency domain. However, to obtain more details of the signal, we employ time-series analysis in the form of spectrograms. A spectrogram is a visual representation of the spectrum of frequencies of a signal as it varies with time. Additionally, it captures amplitude variations in the signal. Spectrograms are widely used in the fields of music, speech processing and linguistics. They can be used to identify words phonetically as well as the sounds of various calls of animals. Spectrograms can be obtained by applying the short time Fourier transform (STFT). In STFT, we apply discrete Fourier transform to each specific frame and show the results accordingly.

Table 1: Information provided by the different representations of an audio signal

Representation	Details available	Details missing
Time domain waveform	Amplitude, time	Frequency
Frequency spectrum	Frequency, Amplitude	Time
Spectrogram	Amplitude, frequency, time	--

Table 2: Application-specific dedicated open source toolkits for implementing various audio processing tasks

Task	Toolkits
Automatic speech recognition (ASR)	Hidden Markov model (HMM) based toolkit (HTK)[1], Kaldi[2], CMU Sphinx [3]
Speaker diarization	DiarTK, LIUM
Automatic speaker recognition (ASR) and verification (ASV)	Bob, ALIZE, SIDEKIT
Music information retrieval (MIR)	Essentia, SMIRK
Common tools for audio data analysis	Audacity, Praat

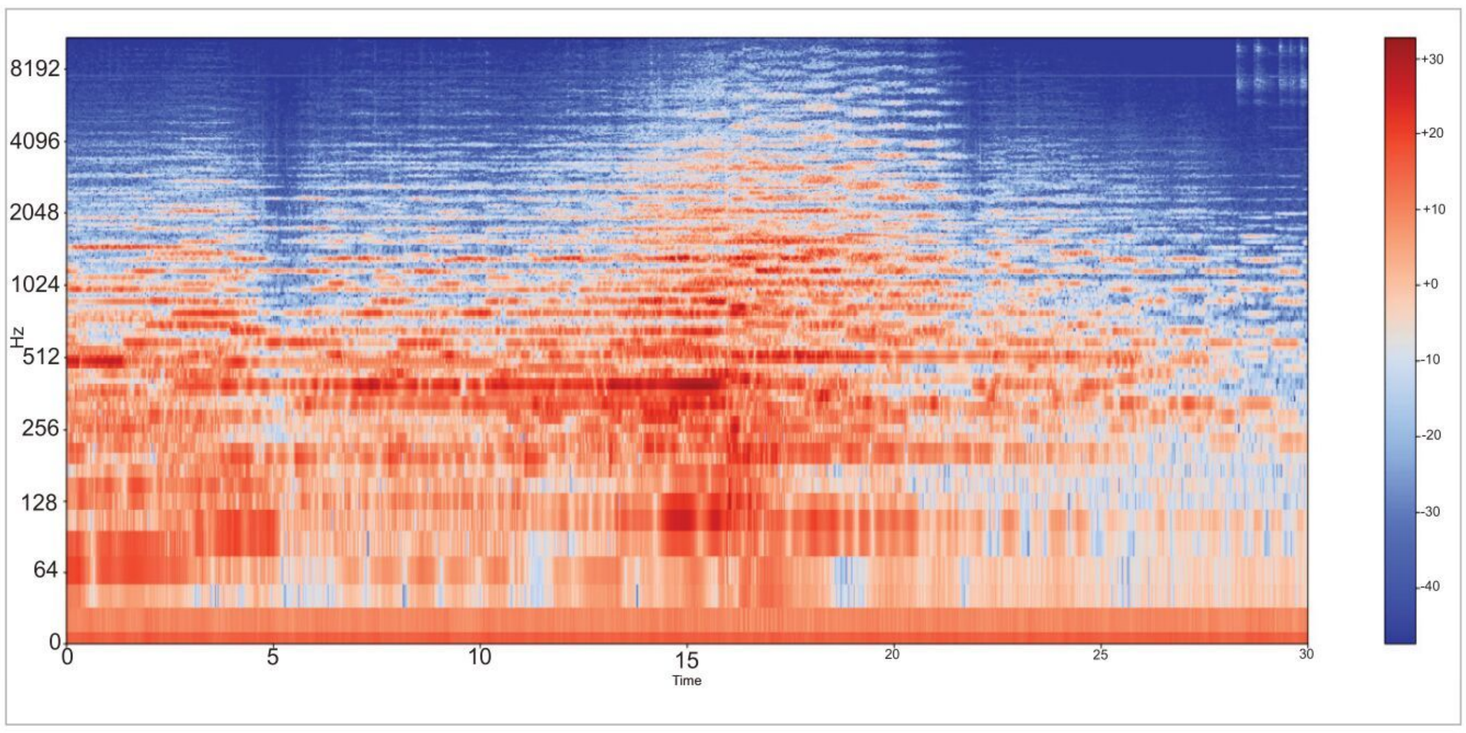


Figure 8: Spectrogram representation of the audio signal

```
def plot_spectrogram(Y, sr, hop_length,
y_axis="linear"):
    plt.figure(figsize=(25, 10))
  librosa.display.specshow(Y,sr=sr,hop_
length=hop_length,X_axis="time",   y_
axis=y_axis)
    plt.colorbar(format="%+2.f")
```

Often, signals are transformed to the frequency domain to reveal the majority of their inherent information. Time domain analysis has certain limitations, which the frequency domain can counter. Both spectrum and spectrograms are frequency domain representations of digital signals. However, in addition to time and frequency information, the latter also provide information about signal amplitude (loudness). Spectrogram based analysis is sometimes referred to as time-frequency analysis.

Voice spectrography analysis is used in crime investigations to match a suspected speaker's voice pattern with that in criminal records. It also helps in applications such as vowel recognition based on frequency components of the signal.

Recent trends

Due to the pandemic, everyone prefers touch-free and hands-free interfaces for interacting with applications. Voice based devices are therefore being used for various applications. Audio spoofing detection is a recent technology, where the system checks whether the voice is generated from a genuine user or is a fake one. An interesting area of research in health care is analysing audios to determine the symptoms of any respiratory disease. In the field of music, speech-music discrimination can be done by removing or subduing the speech parts of the signal, and emphasising only the music.

Now that you have read this article, I am sure you are eager to experiment with audio signals. To begin that journey, just record your voice and play with the code given in this article. You may be amazed at what you discover along the way. END

References

[1] *https://cmusphinx.github.io/wiki/tutorial/*
[2] *http://kaldi-asr.org/doc/about.html*
[3] *https://www.speech.kth.se/~matsb/speech_rec_course_2003/htk_tutorial.pdf*
[4] *https://projets-lium.univ-lemans.fr/spkdiarization/*
[5] *https://alize.univ-avignon.fr/*
[6] *https://www.idiap.ch/software/bob/*
[7] *https://projets-lium.univ-lemans.fr/sidekit/*
[8] *https://essentia.upf.edu/*
[9] *https://smirk.cs.princeton.edu/*
[10] Garg S., Mankad S.H., Voice liveness detection under feature fusion and cross-environment scenario, Multimed Tools Appl 79, 26951–26967 (2020)

By: Dr Sapan H. Mankad

The author is associated with the Department of Computer Science and Engineering, Institute of Technology, Nirma University, Ahmedabad. His areas of interest include audio and speech signal processing, and machine learning. He enjoys writing technical articles and blogs.

Stop Arguing and Start Cashing In on Open Source Specifications

APIs are the building blocks of modern software. Having a contract in place to build an API facilitates easier communication between the parties involved. But what happens when the format of the contract tends to increase arguments among team members? That's where open source specifications that define APIs come in, and spreadsheets don't help.

Open source, in simple terms, is defined as software where the copyright holder grants users the right to use, modify, and distribute the software and its source code to anyone, for any purpose. Despite open source specifications being one of the most critical components of the ecosystem, these are not talked of often. As a matter of fact, the specifications of software are what make it distinctive. These specifications are also discussed, debated on and argued about in development teams, but not much is written about them. Is there a way for teams to stop arguing about, and start cashing in on open source specifications?

Let's answer that question with a story!

The loan app specifications

This story revolves around a team developing an app intended to help bank customers sign up for securing a loan at the click of a button. Sonali, a business analyst in the team, walks up and informs her colleagues that they need to build a feature that allows customers to apply for a loan, using a downstream customer application programming interface (API) exposed by the client. She asks the team about the time required to develop this feature.

Meera, who is an application developer in the team, comes in. She is really smart and does not commit

any timeline for delivering the feature. Instead, she asks and seeks more clarity on the downstream API that will be used. She also asks if there is a sandbox environment to test the API.

Sonali tells Meera that she will loop in the team on the email received from the client. She also explains to Meera about how that email contains documents and spreadsheets relevant to the feature.

Now, as the feature requires a lot of fields (354 to be precise), Meera picks up all the 354 feeds from the sheets, uses her editing skills, and posts the status of the task as 'dev done'. Sonali, after a few days, comes back to meet Meera and informs her that the client has requested for some clarifications about the feature, and the contract. The client, Sonali informs, has also requested for minor changes. Meera leaves the project she is working on, and goes back to the bank loan feature to add and make changes as suggested by the client, but because the changes are minor, she is not quite excited about making them.

Fast forward a few months later, and the story is still not over! The testers who are checking the API realise that there are a lot more things that need to be done. They also notice the error responses that are coming from the API are different from the ones mentioned in the contract. Even the 200 'OK' responses are different, probably because of the way they are defined in the document.

This is the point where Meera gets a little irritated as the work is not interesting. She is also tired of the time the feature is taking, in addition to the blame game taking place inside the team. Hence, at this point, the arguments begin.

A lot of changes later, the team manages to push the feature ahead and, as it gets delivered, the team members start looking at charting a map to zone what did and did not go well.

A few open source specifications that can be used to define your APIs are described briefly below.

OpenAPI Specification (OAS): This specification is useful for defining REST APIs. It is a community-driven open specification within the OpenAPI Initiative, a Linux Foundation collaborative project. This specification defines a standard, programming language-independent interface description for HTTP APIs that allows both humans and computers to discover and understand the capabilities of a service without requiring access to source code.

AsyncAPI: AsyncAPI is an open source initiative that seeks to improve the current state of Event-Driven Architectures (EDA). The long-term goal of the project is to make working with EDAs as easy as it is to work with REST APIs. The AsyncAPI specification settles the base for a greater and better tooling ecosystem for EDAs. You may use this specification if you are looking for a solution to automate and formalise the documentation or code generation of your event-driven (micro)services.

JSON Schema: This allows you to validate JSON documents used in OpenAPI and AsyncAPIs. The specification is split into two parts — core and validation. The core defines the basic foundation and the validation defines the validation keywords of JSON Schema.

What were the troubles?

The first thing the team focused on was the way the contract was sent to them. It found that the contract was defined loosely, and also error-prone. The documents shared by the client made the team go back and forth a lot of times. The business manager had to constantly keep in touch with the client to seek clarification about the pieces that were missing. It turned out that the first major challenge was not coding, but the way the two teams (client and developers) were communicating.

The integration, however, failed even after all the efforts, and the prime reasons for this, the team realised, included incorrectly spelt fields, mismatched data types, case sensitive fields, error parsing issues, missing responses, and more. In a nutshell, the errors arose as proper marking was not done by the client, and the contract was not used in the correct way.

```
RequestError:
     type: object
     properties:
       type:
         description: A URL that uniquely identifies the problem type.
         type: string
         example: "/problem/validation-error"
     title:
       description: A human readable description of the error.
       type: string
       example: "Your request parameters didn't validate"
     status:
       description: The HTTP response code.
       type: number
       example: 400
     instance:
       description: The URL of the call for which the error occurred.
       type: string
       example: "/media/analyze"
     detail:
       description: An optional field with more information depending on the type.
       type: string
       example: "job_id is required"
```

Figure 1: Definition of request error

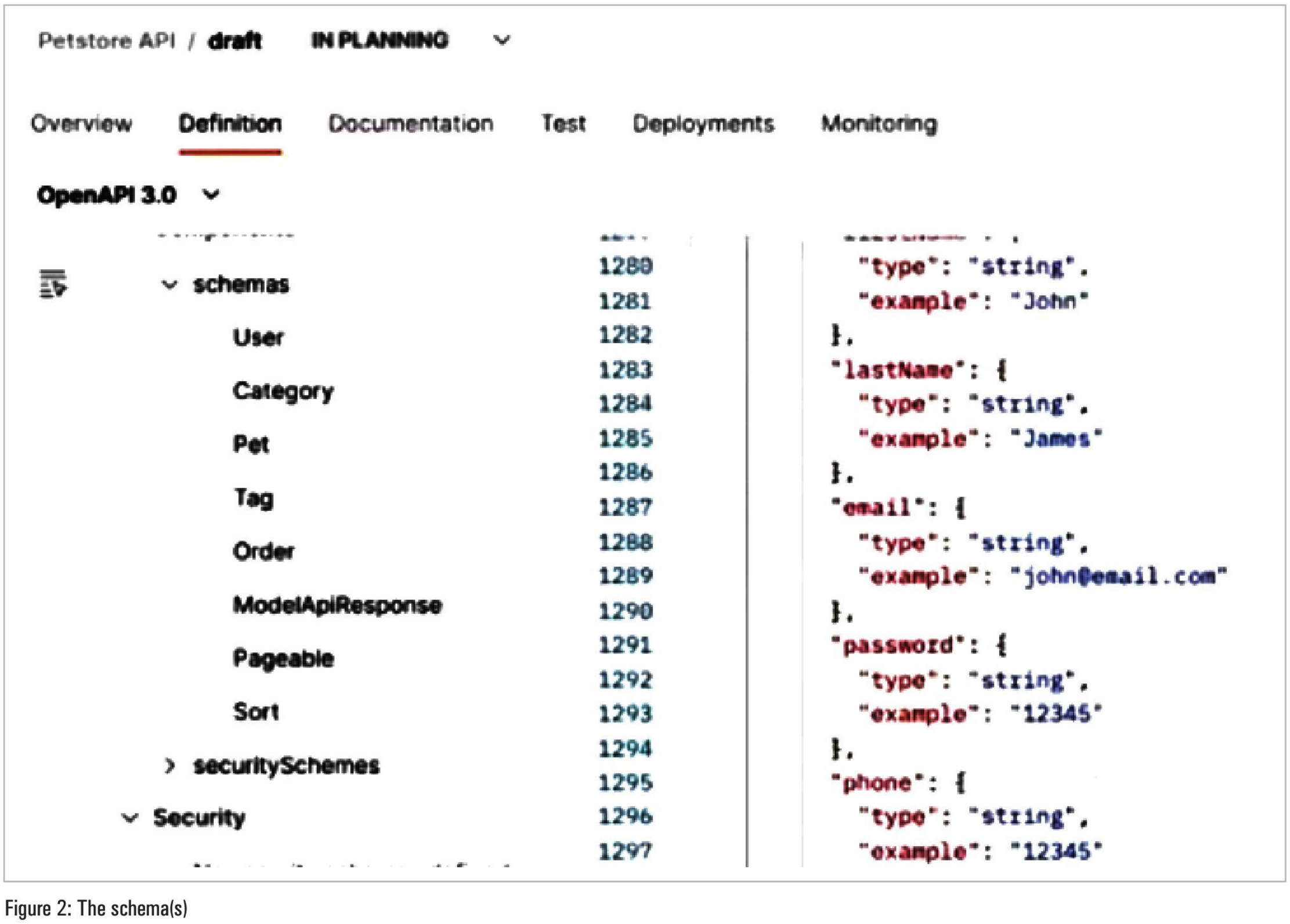


Figure 2: The schema(s)

The way to specifications

The journey to deliver better APIs using open source starts with "communicating APIs via specifications." If the team developing a feature (or features) is able to understand the specifications better, it gets into a good position not only to develop, but also pass the requests to the entire community of developers better!

Choosing specifications over a doc is the first choice developer teams need to make. Of course, a spreadsheet might look more tempting to deploy as it doesn't come with a lot of rules, but a specification can offer the same benefits and a lot more, such as conformance to a format. It's important to conform to a format, as it helps to choose one that is similar to an API. Spreadsheets, on the other hand, contain plain text. A format, whether in JSON/YAML, or anything else, is much closer to an API than plain text.

Next comes the clarity derived from schema objects, which are basically defined as logical structures created by users. Objects such as tables or indexes, which hold data in many cases, also consist of a definition. In the Dolby Media Analyze OpenAPI, RequestError is a schema object defined as shown in Figure 1.

Schema objects can be defined, and this will resolve the error response parsing issues the team faces. Figure 2 shows an example of a defined schema object. Schema validations on the specification help ensure at least one successful response is defined for a path. There can be a lot of other validations on the schema too. Tooling like documentation, code generation, mock servers can be used to support the specifications further.

These specifications, in short, act as a blueprint for APIs, which can be of great help in describing the overall structure of an API.

"Spreadsheets served us well in the information age, but as enterprises wrestle with their digital transformation, they are realising they need a much more precise set of human and machine-readable tools that can track the exponential increase in data being produced each day. These specifications now help teams keep up with the perpetual pace of change and remain competitive in today's ever-evolving and shifting digital landscape," says Postman chief evangelist Kin Lane. END

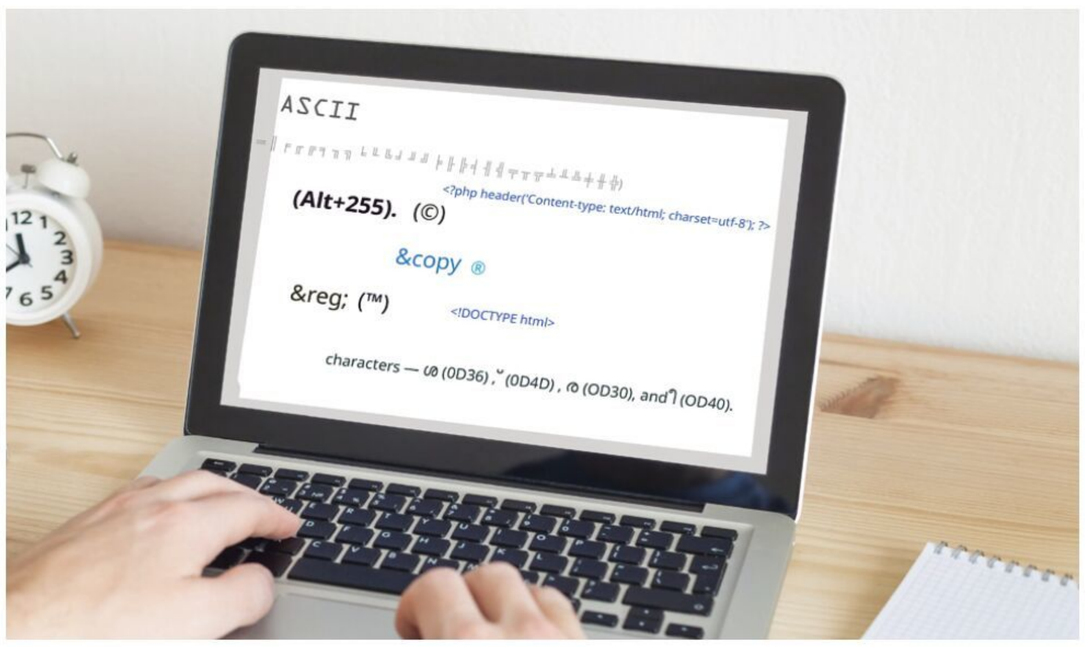
This article is based on a talk given by Meenakshi Dhanani, developer advocate, Postman, at OSI 2021.

By: Mukul Yudhveer Singh

The author is business editor at *OSFY*.

Try These Unusual Unicode Tricks

Much thought from the finest brains all over the world has gone into the making of Unicode. Its main purpose is to serve all mankind. If you are a developer, do not make the mistake of ignoring this encoding standard.



Some things that nobody teaches us are learnt the hard way. Did your computer course teach you to acquire resources as late as possible and release them early? Did any programming book tell you that software should not be developed using Visual Studio while it is running with administrator privileges? Every developer goes through the process of learning new and best-practices while unlearning old and die-hard ones. If you haven't read the article titled, *The Absolute Minimum Every Software Developer Absolutely, Positively Must Know About Unicode and Character Sets (No Excuses!)* written in 2003 by Joel Spolsky (co-founder of *StackOverFlow.com*) , I suggest you read it before writing another line of code.

Unicode

Unicode is a unified text scheme for representing characters of over 150 languages, the list of which also includes Braille, sign language, Esperanto, musical notations, hieroglyphs, cuneiform, chemical symbols, emoticons and dingbats. Microsoft had a big font named *Arial Unicode* MS that was supposed to support all of Unicode. However, the tech giant gave that up, once Unicode continued to grow.

When learning programming for the first time with a language such as C, it is easy to assume that a character is the same as a byte or that a byte comprises only 8-bits. If your computer course rushed through such details, please run through and spend some time on the basics from a proper C language book. Spolsky's article mentions how the creators of PHP initially wrote the Web scripting language without support for Unicode!

ASCII

Unicode represents a paradigm shift from the days when boxes were drawn in the headers of C code using characters in the extended ASCII set (128 extra codes added to the 128-code ASCII). It is better than traditional ASCII art even today.

In Unicode, the box-drawing characters (╞ ╡ ╤ ╧ ╪ ╫ ╬ ═ ╒ ╓ ╔ ╕ ╖ ╗ ╘ ╙ ╚ ╛ ╜ ╝ ╟ ╠ ╢ ╣ ╥ ╦ ╨ ╩ ╫ ╬) were moved further up into the stratosphere.

In the early days of computing, I worked primarily on DOS. Windows was a GUI program that ran on it. On a DOS keyboard, you could type the copyright symbol (©) by holding down the *Alt* key and typing *0169* on the numeric keypad. On lab computers that did not have floppy drives, I created undeletable directories in the hard disk by suffixing the directory names with an undetectable space symbol (*Alt+255*). A more subversive trick was to use memory utilities and change the name of a directory to 'CON' in the file allocation table (FAT). Windows, too, would not allow you to touch a file or directory named CON because it was a reserved file descriptor for the console in DOS. For many years, Linux would let you create a directory named CON in a Windows partition if you wanted to. But that was fixed later.

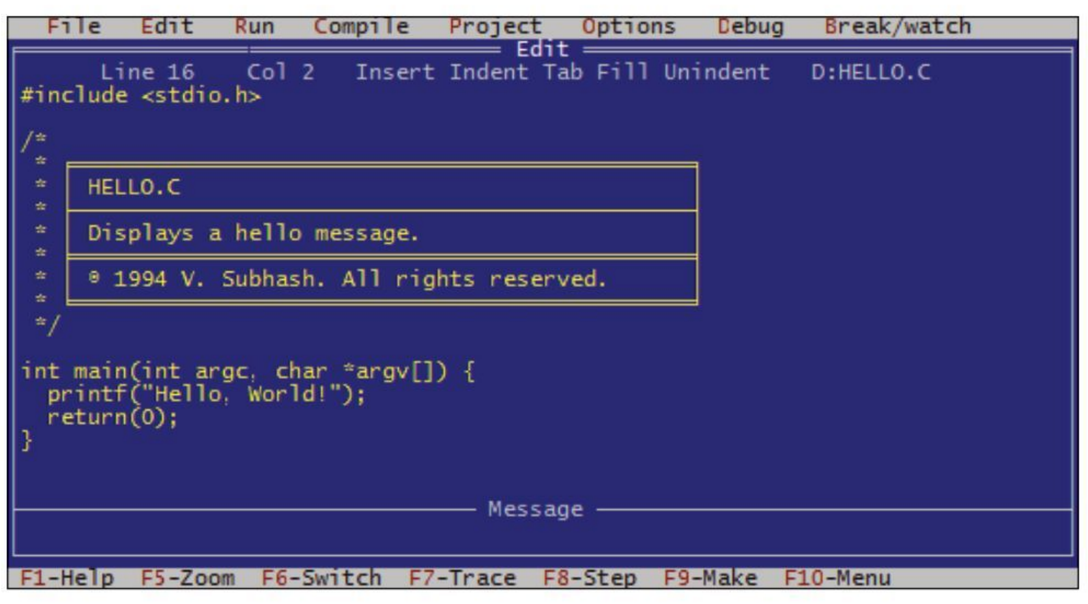


Figure 1: The uber way of beautifying comments in source code

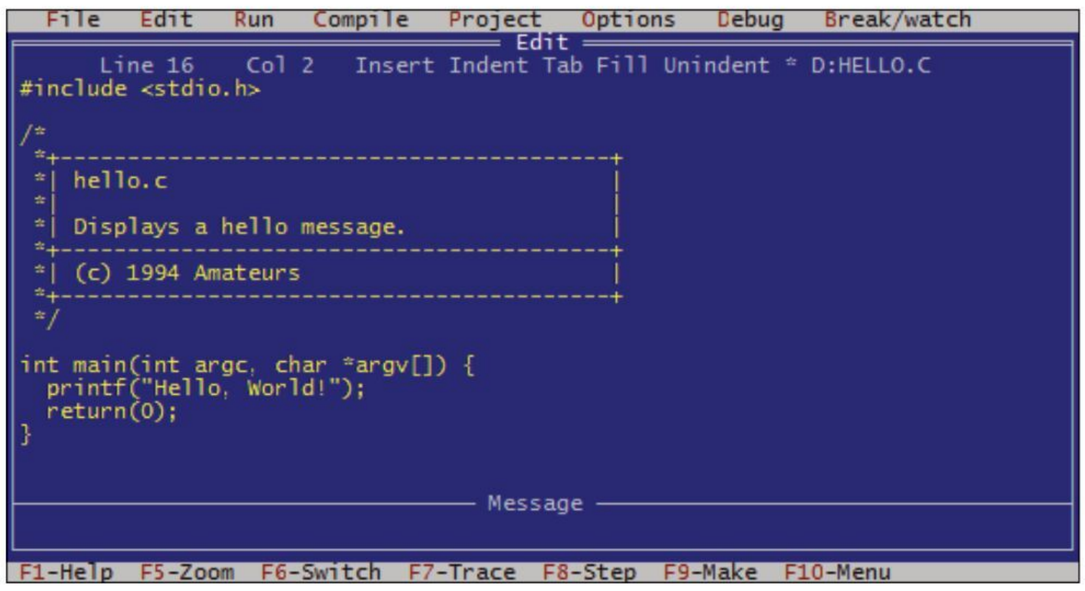


Figure 2: The spartan way of highlighting comments in source code

HTML entity references

When I started learning HTML, I found that the copyright symbol could be written as *©*. The registered symbol (®) could be written as *®* and Trademark symbol (™) could be written as *&trade*. HTML has several such character entity references. However, any character can be written using its Unicode codepoint, which is known as a numeric entity reference. For the copyright symbol, this is *©*. It can also be written as *©* with the *x* signifying that the codepoint is hexadecimal. Similarly, the numeric entity reference for the registered symbol is written as *®* or as *®*.

Source code file encoding

If your OS is Windows, then it is likely that you save all your source code files in the default Windows encoding *Latin-1* or *Western European*. These do not work well with non-English Unicode strings. (Ideally, you should be storing all UI strings in a resource file so that file encoding is never a problem.) However, for maximum resilience and portability, save all your source files in UTF-8 encoding. If your file turns into gibberish, you need to first select everything in the file to the system clipboard (*Ctrl+A* and *Ctrl+C*), then change the encoding, and finally do a paste (*Ctrl+V*). If your editor allows you to change the encoding in the 'Save as' dialog box, then paste the text after saving the file. With UTF-8 encoding, you will eliminate a whole heap of trouble.

HTML forms

If you have a Web script that outputs HTML to a browser, then let the browser know the encoding of the output stream. In PHP, the first line could be:

```
<?php header('Content-type: text/html; charset=utf-8'); ?>
```

This is not enough. In the HTML, declare the encoding as early as possible:

```
<!DOCTYPE html>
<html>
 <head>
   <meta http-equiv="Content-Type" content="text/html;
charset=UTF-8" />
   <title>This page is Unicode-encoded</title>
   …
```

This is particularly important if you have HTML forms where you accept text data. If your HTML page is not Unicode-encoded or if the server-side script is not saved with Unicode encoding, then you may not always be able to correctly process non-English text input.

Fonts

Despite the above, if you still get the dreaded question marks or boxy characters, it means you do not have the font to render the non-English text. The Unicode project only specifies numbered codepoints for its characters. The actual visual representation of the characters is left to the font engine and fonts installed in the output device. For example, in the Malayalam letter 'Shree', there are four Unicode characters – ശ (0D36), ◌ീ (0D4D), ര (OD30), and ◌ു (OD40). A font that supports Malayalam decides how to combine them as ശ്രീ. You will not find the finally rendered ligature in Unicode. If you move the cursor across the letter, you will be able to split it into two ligatures ശ് and രീ. Even these will not be found in Unicode. How they are formed is left to the font maker.

You will have to view a ligature in a hex editor to identify the individual codepoints. Then, you can search the codepoints in the Unicode list. That will tell you what language the character belongs to. You can then install a font that supports that language.

The full character list is published on the Unicode website at *http://ftp.unicode.org/Public/UNIDATA/UnicodeData.txt*. This file tips the scale at nearly 2MB. It is in the form of a comma-separated file (CSV). With some JavaScript (*http://www.vsubhash.in/refs/unicode.html*), you can render it as a table. (The following code will tax the JavaScript and HTML engines of a browser. Do not run it on a system that is light on resources.)

```
<!doctype html>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html;
charset=UTF-8" />
    <link href="universal.css" rel="stylesheet">

<script>

// Make a copy of the Unicode data text file in the
```

```
// directory of this HTML file.
var sUnicodeURL = “UnicodeData.txt”;
var sUnicode=””;

var oXhr = new XMLHttpRequest();
oXhr.addEventListener(“load”, function() {
                              sUnicode = oXhr.responseText;
                              processUnicode();
                             });
oXhr.open(“GET”, sUnicodeURL);
oXhr.send();


function processUnicode() {
  var arLines = sUnicode.split(‘\n’);
  var n = arLines.length-6;
  console.log(“Lines = “ + n);

  var oTable = document.getElementById(“tbl_unicode”);

  var sTableHTML = “<tr><th style=\”width: 6em;
\”>Codepoint</th><th  style=\”width: 6em; \”>Hexadecimal</
th><th>Description</th></tr>”;
  for (var i = 0; i < n; i++) {
    var arUnicodeData = arLines[i].split(‘;’);

    var sDescription = “”;
    var sCode = arUnicodeData[0];

    if (arUnicodeData[1] == “<control>”) {
      sDescription = “Control Character”;
    } else {
      sDescription = “”;
    }

    if (arUnicodeData[10]) {
      sDescription = sDescription + arUnicodeData[1] + ‘ (‘ +
arUnicodeData[10] + ‘)’;
    } else {
      sDescription = sDescription + arUnicodeData[1];
    }

    sTableHTML = sTableHTML + “<tr><th>&#x” + sCode + “;</
th><td>” + sCode + “</td><td>” + sDescription + “</td></tr>”;

  }
  oTable.innerHTML = sTableHTML;
}
</script>

  </head>
  <body>

<table id=”tbl_unicode”>
</table>

  </body>
</html>
```

This table is much easier to browse than the *Character Map* application in Linux.

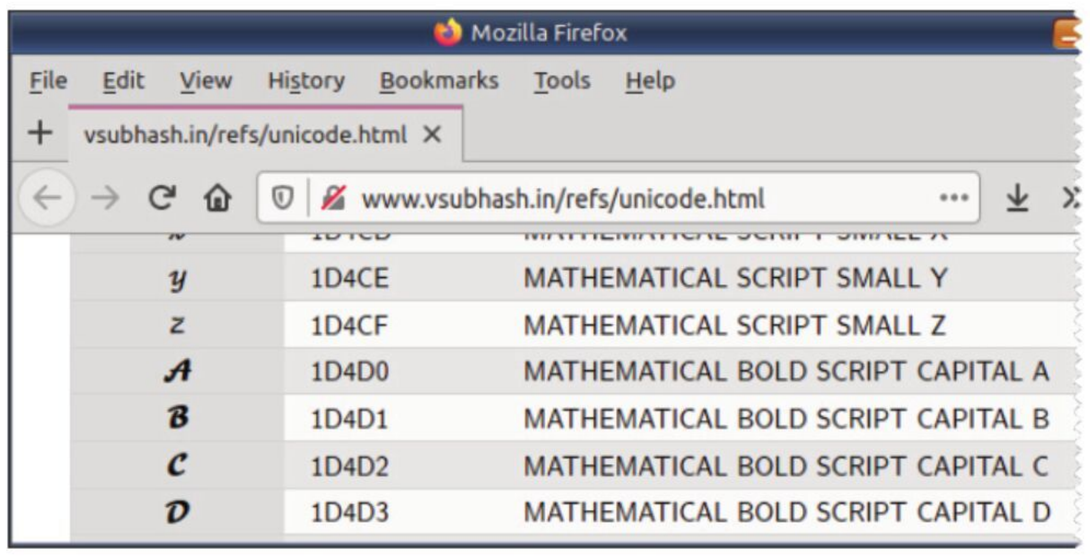


Figure 3: This table shows part of the transformed Unicode data list

With this table, I can easily write my name as ‘𝓥. 𝓢𝓾𝓫𝓱𝓪𝓼𝓱’ in my email. This name stands out in any email inbox. However, the disadvantage is that no one will find it by searching for ‘V. Subhash’. (*Disclosure*: I learnt this from YouTube bootleggers, who use the technique to escape filters. Interestingly, You Tube search understands this technique.) With Unicode and without any images, my name can be written in different ways, as shown in Figure 4.

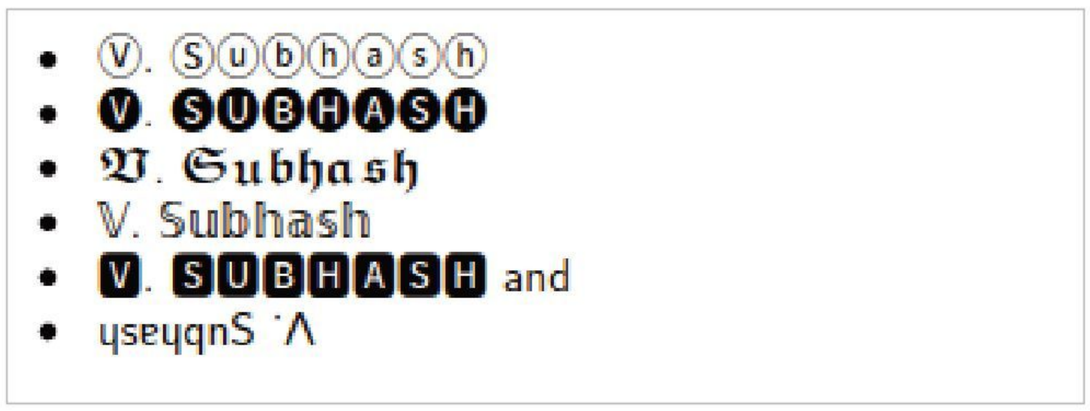


Figure 4: Different ways of writing names

Of course, Windows is always late to the game and may not have full font support for such Unicode strings. Linux has no such problem.

Twitter has been blamed by many for ruining the Internet for them. (Twitter was one of the first to withdraw RSS feeds. Can you really blame it, though? Even Firefox has removed RSS support!) It was a surprise when Twitter created a free and open source emoticon-heavy font that anyone can download and use from *https://github.com/twitter/twemoji*.

Universal fonts style sheet

When you have a website, the text in the pages is displayed

Figure 5: The open source Twitter font has lots of colour emoticons. This one, though, did not go down well with the thought police.

using the fonts on the end user's device. The best approach is to use whatever fonts that are available. Ideally, merely mentioning *serif*, *sans-serif* and *monospace* in the CSS should be fine but browsers typically choose fonts from very ancient times for the default options. I suggest you use a universal font CSS (stylesheet) like this so that newer OS-optimised fonts are chosen instead.

```
body {
  /* Order: Special,
   *        Android, iOS,
   *        Linux (Liberation, Free, DejaVu),
   *        Mac UI, Windows Vista UI, Windows XP UI,
   *        Mac Unicode fallback, Windows Unicode fallback,
   *        Adobe Standard Type 1, generic
   */
  font-family: “CMU Sans Serif”,
               “Roboto”, “San Francisco”, “Helvetica Neue”,
               “Liberation Sans”, FreeSans, “DejaVu Sans”,
               “Segoe UI”, Tahoma, “Lucida Sans Unicode”,
               “Last Resort”, “Arial Unicode MS”,
               Helvetica, sans-serif;
}

h1, h2, h3, h4, h5, h6 {
  /* Order: Special,
   *        Android, iOS,
   *        Linux (Liberation, Free, DejaVu),
   *        Mac, Windows Vista Serif, Windows XP Serif,
   *        Windows Unicode fallback, Mac Unicode Fallback,
   *        Adobe Standard Type 1, generic;
   */
  font-family: “CMU Serif”,
               “Roboto Slab”,
               “Liberation Serif”, “FreeSerif”, “DejaVu
Serif”,
               Times, Constantia, “Trebuchet MS”, “Times New
Roman”,
               “Lucida Sans Unicode”, “Arial Unicode MS”,
               “Last Resort”,
               Roman, serif;
}

code {
  /* Order:    Special,
   *           Android, iOS,
   *           Linux (Liberation, Free, DejaVu),
   *           Mac, Vista, XP,
   *           Adobe Standard Type 1, generic
   */
  font: normal 1em “Source Code Pro”,
                   “Roboto Mono”, Menlo,
                   “Liberation Mono”, FreeMono, “DejaVu
Mono”,
                   Monaco, Consolas, “Lucida Console”,
“Courier New”,
                   Courier, monospace;
}
```

The corporate control-freak approach is to use a proprietary font with a *@font-face rule*. (This approach forces the browser to redraw the text after the font file is downloaded.)

```
@font-face {
  font-family: ‘CNN’;
  font-style: normal;
  font-weight: 400;
  src: url(/fonts/cnn.ttf);
}

body { font-family: CNN, Helvetica, san-serif; }
```

The cheaper control-freak approach is to use an obscure font from the Google fonts CDN with a *LINK HTML* tag. (This CDN is actually very slow and forces the browser to redraw the text after a long delay. Until then, large sections of the page will appear blank. Do read my 2019 CodeProject article titled ‘How to Make Your Website Serve Pages Faster’ for more information.)

```
<head>
  <link rel=”stylesheet” href=”https://fonts.googleapis.com/
css?family=Gentium”>
  <style>
    body {
      font-family: “Gentium”, Roman, serif;
    }
  </style>
</head>
```

The Linux Unicode way

In Windows, you can use *Alt+number* pad key combinations to type Unicode characters. In Linux, you need to first press *Ctrl+Shift+U*. The cursor will temporarily transform to an underlined *u*. Then, when you type the codepoint number and press the *Enter* key, the corresponding Unicode character will be inserted at the text cursor. To type the copyright symbol (©), you need to type *Ctrl+Shift+U*, *a9*, and then *Enter*. For the Indian rupee symbol (₹), you need to type *Ctrl+Shift+U*, *20B9*, and then *Enter*.

Linux offers an additional way of typing some characters using a 'Compose' key. This is a modifier key that is designated as such in the keyboard settings. On my computer, I have set the useless *Windows* key as its *Compose* key *(Preferences » Keyboard » Layouts » Options » Compose key position » Left Win)*. Now, to type the copyright symbol, I type *Windows+O* (simultaneously) and then *c* (subsequently).

The trouble with space

In my first year of self-publishing (2020-21), I created 21 books. I wrote, edited, illustrated, designed and formatted them myself — all thanks to open source software. For every book, I built a PDF file for the paperback (using a shell script) and an EPUB file for the ebook (manually). I wrote the manuscript in the text-only CommonMark format. (This is a new standardised dialect of the old MarkDown format.) I used the CommonMark executable to convert the text manuscript to HTML. Then, I styled the HTML with some CSS. This stylised HTML was fed to KhtmlToPDF to create the paperback PDF file. (I used Calibre to create the ebook copy from the same HTML.) However, the PDF output had a few problems.

The current KhtmlToPDF was resurrected by an Indian from an almost-abandoned open source project. KhtmlToPDF uses a headless Firefox browser of considerable vintage. It prints (not displays) the input HTML page. (KhtmlToPDF provides plenty of options for controlling the pagination and writing auto text such as page numbers, headers and footers, before exporting the print output to PDF using iText.) The printing talents of the Firefox browser are good but not perfect. Randomly, the space after an italicised word would disappear. If I manually added an extra space, it would be too much and the blank space would stand out. Initially, a fix seemed hopeless. But then, Unicode came to my rescue.

Key	Description	First key press	Second key press
©	Copyright	⌨ o	c
®	Registered	⌨ o	r
™	Trademark	⌨ t	m
€	Euro	⌨ =	e
£	Sterling pound	⌨ =	l
á	'a with accent' as in *á la*	⌨ '	a
œ	'a with e' as in *amœba*	⌨ o	e
½	half	⌨ 1	2
⅙	one-sixth	⌨ 1	6
☺	smiley face	⌨ :	)
☹	frowning face	⌨ :	(

Figure 6: This table shows the Compose key as ⌨ . You will find more comprehensive lists of Linux compose-key combinations online. I have included a big one in my book *Linux Command-Line Tips & Tricks.*

Codepoint	Description
2002	EN SPACE
2003	EM SPACE
2004	THREE-PER-EM SPACE
2005	FOUR-PER-EM SPACE
2006	SIX-PER-EM SPACE
2007	FIGURE SPACE
2008	PUNCTUATION SPACE
2009	THIN SPACE
200A	HAIR SPACE
...	...

Figure 7: These are a few of several types of space characters in Unicode

Unicode offers several types of space characters. There is space about the size of an 'n' — the en space, and also a space about the size of an 'm' — the em space, both of which may seem familiar if you know CSS. But, that is not all. I experimented with several other Unicode space characters and, eventually, I could hide KhtmlToPDF's deficiencies.

In some places, the last letter in a line would be partially displayed. The fix for that was to use the non-breaking space character (*Compose + Space + Space*) after the word, and it would conveniently wrap to the next line. Sometimes, I did not want a line to break at a hyphen. Conveniently, Unicode has a non-breaking hyphen (*Ctrl+Shift+u* and *2011*) as a fix.

That's not all

You can already see that I am now addicted to the long hyphen (—) but you may not have noticed that instead of three dots (...), I have used an ellipsis (…). This is just one Unicode character, instead of three. I do not use apostrophes for my quotations. There are dedicated Unicode characters for it. I type them with the *Compose* key. I type 'Hi' instead of 'Hi'. I write "Hello, World", instead of "Hello, World". But, not when I write code. When other writers write source code in Microsoft Word, the annoying software autocorrects all apostrophes and double quotation marks into Unicode quotation marks. Such a code listing will not compile in an IDE. It becomes a nightmare for the person who does the page setting. My manuscripts (including the code listings) are entirely in text (as CommonMark). What is written in text, stays in text. So whether it is exported to HTML, Word or PDF, the code will always compile. END

By: V. Subhash

The author has written 22 books including *FFmpeg Quick Hacks (free), CommonMark Ready Reference, Linux Command-Line Tips & Tricks, Cool Electronic Projects* and *How To Install Solar.*

Do You Have a Strategy to Manage Open Source Risks in Web Applications?

As per an IDC report, every organisation is a software company today. Staying competitive depends on how fast your business can transform, and embrace new tools and technology. For high-performing organisations, this means open source software is now a matter of 'when', and not 'if' it should be employed. But there are risks involved in the use of vulnerable open source software components which, thankfully, can be managed easily with the right strategy.

Use of open source components has become an integral part of software development today. A key advantage is the significant reduction in cost and time of development, which frees up engineers to focus on deeper challenges of their products while accelerating the time to market. With the Covid-led digital transformation of businesses, the adoption of open source software is going to increase all the more.

While use of open source software brings benefits and opportunities, it has risks and challenges too. An important challenge that comes along is the security of these components and, eventually, the security of the products that make use of them. For example, a new critical vulnerability, named Log4Shell, was recently disclosed in a few versions of the worldwide popular Log4j library exposing many applications to potential attacks.

The widespread use of open source components requires that we take a more proactive approach to open source management to make sure the software products that we are building do not contain vulnerable components. As these security vulnerabilities are disclosed publicly, they are easily exploitable. The OWASP's Top Ten list of Web application security risks was updated in 2017 to cover this risk; it has now moved up the list and is referred to as 'Vulnerable and Outdated Components'.

Managing open source risk in applications

One might say that the easiest way to reduce the risk is to avoid using components created by others. This could be the safest option, but it's quite unrealistic today as it deprives one of the large pool of ready-to-use components that can foster the speed of development, which is paramount to stay competitive.

In view of this, it is important to implement a good open source strategy to set and enforce policies that are automated throughout the SDLC. Tools and methods should be put in place that can automatically discover open source dependencies (direct as well as transitive) in applications, cross-check them against published vulnerabilities, trigger alerts when risks are identified and even block the release!

Elements of an open source management programme

A few key elements for building effective processes and strategies for safe consumption of open source are shared below.

Avoid using vulnerable components: In many places, developers are free to choose the open source components and integrate them in their applications. In doing so, what often gets ignored is whether the particular version of the chosen component has any reported vulnerabilities. A suggested way to safeguard applications is to prevent using vulnerable components right from the start when building new products. Therefore:

a) It is important that when you pull new dependencies, pick them only from official sources through secure channels.
b) Check online resources to see if there are any published vulnerabilities. However, the information about vulnerabilities might be hard to find at one place, which makes it difficult for developers to detect components with flaws.
c) One way of ensuring the above and regulating the undesired usage could be to disable the direct access to public repositories and build your private central repository of only approved components. The applications can pull dependencies only from this private repo, and any new dependency addition must go through a security review and approval process.

Identify affected components: There is a possibility that a component was carefully chosen, but a vulnerability got identified and reported sometime later. Also, there would be existing and legacy applications that were built when this system was not in place. So, applications must be continuously scanned to know if the open source components that are already a part of your inventory and products contain any vulnerabilities. This can be automated through the use of various commercial or open source tools. Many of these support a wide range of languages and help not just identify but also categorise the vulnerabilities as per the severity.

Prioritise and fix: With thousands of new vulnerabilities disclosed every year, the scan report will give you a long list of issues to fix, especially when you scan your projects for the first time. You need to understand the severity of vulnerabilities to see which ones are the most critical for you. Many tools help classify issues using established methods and parameters like CVSS, CWE, etc. Issues that have high severity scores, easy exploitability and an available fix could be addressed first. Besides this, one must factor in the business impact of a vulnerability such as the possibility of an SQL injection, any kind of financial data threat, PII (personally identifiable information) leakage or IP loss, and prioritise accordingly. Each application is different and so is the associated risk.

As these are publicly disclosed vulnerabilities, the fixes usually involve an upgrade to a higher version of the dependency. They will be easier to fix if upgrades don't break the existing functionality, but sometimes there could be a major version upgrade which might require bigger changes. Similarly, some fixes may require customers to download a new release, and you may not want to bother customers with too frequent updates. And sometimes the affected components might no longer be supported or have reached end of life, which will require switching to an alternative component that is actively developed or build one in-house. Taking all of this into account, one can prioritise in a way that there is balance between the pace of development and protection against critical issues.

Build pipeline integration — shift left: Implementing measures at the right stage of the product development life cycle is important too. Finding and fixing at a very late stage can cause significant delays in release and increase the project cost. Most of us already have CI/CD pipelines in place to automate the build, test and deployment process. These pipelines can be leveraged to plug vulnerability detection and accelerate automated scanning, find issues early and often, and control the process from a single place. Build-wise scan reports are also available, and builds with critical vulnerabilities can be flagged or even blocked. There can be IDE integrated scans, too, to get an early feedback.

Continuous monitoring: While some projects remain in active development and get continually scanned at the CI/CD security gate, those less updated might have latent vulnerabilities that remain unidentified for a long time. By having a process to run periodic scans on repositories we can constantly check for new vulnerabilities and contain the risk.

By implementing a good open source management strategy, which ensures continuous monitoring and remediation that pervades the entire development life cycle, we can optimise the benefits of open source and accelerate software development while minimising the potential risks. END

References

[1] *https://resources.github.com/whitepapers/Open-source-enterprise-innovation/*
[2] *https://owasp.org/www-project-top-ten/*

By: Minni Arora

The author works as associate vice president architect with Naukri.com. She has fourteen years of experience in architecture, design and development of Web and mobile applications.

When is NoSQL the Best Choice for Database Management?

NoSQL databases can handle unstructured data. Find out how they differ from other databases, and when and where they should be used.

Different types of databases can be used for managing structured and unstructured data. Relational databases are used for structured data, which is formatted and has well-known types. NoSQL databases are used for unstructured data. A database management system typically has an interface, a query processing engine, metadata processor, indexing engine, and storage manager. These components help in querying and storing data into the DBMS. Earlier, blobs (an acronym for binary large objects) were used for storing unstructured data. Tabular data stores evolved to columnar data stores to handle data that is not normalised.

NoSQL databases differ from an SQL DBMS in the way data is available — unstructured or structured. SQL is the query language for relational databases. These have well defined schema. Relational databases are used in transactional or reporting scenarios. NoSQL databases are used to handle unstructured databases in data lake scenarios. The data can be document based and key value type, or can be graph based. Fields can be added dynamically, just like in a document.

The most popular NoSQL databases are MongoDB, Apache CouchDB, Apache HBase, Oracle NoSQL, Apache Cassandra, Neo4j, BigTable, Riak, Objectivity Infinite Graph, and MariaDB. Many enterprises are choosing NoSQL databases to handle data from multi-channel and IoT apps.

Data types

NoSQL database management systems were developed to handle key value data types, wide column data types, documents, and graph data types. A key-value data store is based on dictionary/hash tables. Document data stores can handle data formats like XML, YAML, and JSON. Graph data stores can handle graph based data. NoSQL databases are performant and scalable. REST, Graph QL, and other NoSQL query methods are used for accessing data from NoSQL databases.

SQL DBMSs handle data types like int, char, varchar, binary, varbinary, float, decimal, double, bit, blob, bool, longtext, tinytext, mediumtext, and text. Relational data format is used for transactional data processing like commerce ordering, accounting, and payment systems.

Programming language support

A structured query language is used for creation, updation, modification, and deletion of schema and data. In the design phase, a relational database (SQL DBMS) is designed for its schema and database procedures. NoSQL DBMS is based on a dynamic schema for handling unstructured and semi-structured data. Columns can be

added dynamically and the syntax is different across the NoSQL databases. The query languages for NoSQL can handle documents, columns, graphs, and key-value data types.

Programming languages like Java, C#, Python and others provide support through JDBC drivers for SQL databases. Similarly, they provide support for NoSQL data access.

Query languages

> "The cost of managing traditional databases is high. Mistakes made during routine maintenance are responsible for 80 per cent of application downtime."
> *—Dev Ittycheria, president and CEO of MongoDB Inc.*

An important design concern is the query performance. Query languages need to have support for optimisation to handle complex queries. SQL provides stored procedures and constructs to handle complex data processing programs. It is a lightweight and declarative query language. In NoSQL, query languages are not optimised for handling Big Data. On the other hand, a NoSQL database is scalable and flexible. Big Data frameworks such as Hadoop MapReduce and others help in handling complex data processing for a NoSQL database. In Web applications, developers handle the complex query processing in the application layer, and keep the data access layer plain and simple.

Performance

NoSQL database management systems can be used for creating documents, adding columns to the schema, storing documents, handling complex structures, and content. NoSQL databases abide by the BASE model — basic availability, soft state, and eventual consistency. They ensure data is available, and handle the changes in the state of this data. A NoSQL DBMS ensures the consistency of data. SQL databases abide by ACID (atomicity, consistency, isolation, and durability) properties. A few NoSQL databases also do so.

SQL is used for data querying and modification. The query performance can be optimised easily in SQL databases. In NoSQL database based architectures, query processing is handled in the business layer and not in the data access layer.

Scalability

Relational DBMSs are scalable, and load can be handled by adding more processing power, memory and storage. NoSQL DBMSs can be scaled by adding new database nodes to handle the traffic using sharding. Sharding helps in handling Big Data and data which is changing.

Huge volumes of data can be processed at higher velocity by scaling out. Unstructured/semi-structured data can be stored by using the NoSQL datastore. This type of DBMS is consistent, performant and scalable. NoSQL, by definition, is not SQL-compliant. It has its own query language to handle data processing. Vertical scaling can be visualised as adding more computing power vertically. Horizontal scaling is like growing a farm by adding more devices horizontally.

ACID properties

Tables are used for designing database schema in SQL databases. NoSQL database management systems use key-value data types, documents, graphs, and wide column data types. SQL DBMSs abide by atomicity, consistency, isolation, and durability (ACID) properties. ACID properties help in preserving the integrity and validity of data. NoSQL DBMSs are based on the CAP theorem, which was coined by Brewers to focus on consistency, availability, and partition tolerance properties. This theorem states that a distributed database system can only have two of the three properties of consistency, availability and partition tolerance.

Hence an SQL DBMS is suitable for transactional applications like online ordering systems. NoSQL is good to handle unstructured data.

Convergence of NoSQL and SQL DBMSs

There are a few NoSQL DBMSs that comply with ACID properties as well as the CAP theorem, signifying a convergence of the SQL and NoSQL DBMSs. MySQL and other SQL databases provide a document store to handle unstructured data. MongoDB, on the other hand, provides features to handle ACID based transactions. Similarly, AWS managed NoSQL, DynamoDB, and others abide by ACID properties.

What's next?

> "The choice to use a NoSQL database is often based on hype, or a wrong assumption that relational databases cannot perform as well as a NoSQL database. Operational costs, as well as other stability and maturity concerns, are often overlooked by engineers when it comes to selecting a database."
> *—Yoav Abrahami, Scaling to 100M: MySQL is a Better NoSQL | Wix Engineering*

Both SQL and NoSQL DBMSs come with their own advantages and disadvantages. If one DBMS can handle relational and unstructured data, it works the best in terms of cost and operational efficiency. END

By: Bhagvan Kommadi

The author is founder of 'Quantica Computacao', India's first quantum computing startup. He has 20 years of experience in product development and IT services.

CODE SPORT

Sandya Mannarswamy

In this month's column, we discuss entity disambiguation, which is an important problem in natural language processing.

Welcome to the New Year 2022. It seems like 2021 went by in a whirl with the pandemic cloud hanging over all of us. Hopefully, 2022 will be better as we learn to cope with the changes brought on by the pandemic. We had discussed a number of computer science interview questions in the last column. In this column, we take a break from interview questions and instead focus on a popular natural language processing (NLP) topic, namely, entity disambiguation.

One of the questions we discussed in last month's column was on entity disambiguation. The latter is important for relation extraction. Entity definitions are unambiguous entity concepts typically defined in a structured knowledge store such as a knowledge base or knowledge graph. There are a number of structured knowledge bases such as Freebase, YAGO or DBPedia. These knowledge bases contain relations between entities typically expressed as a triple <Barack Obama, President Of, United States> with the two involved entities being <Barack Obama> and <United States>, and the relation between them being <President of>. While knowledge bases like Freebase can provide structured knowledge representations, the amount of knowledge represented in them is limited and static. Hence, AI/NLP systems often look at extracting knowledge from unstructured text in addition to these structured knowledge sources. Wikipedia is a common source of unstructured knowledge, and is often mined for extracting relations from it automatically to populate structured knowledge sources.

Relation extraction is often a challenging task. Unstructured text contains entity mentions that need to be identified as entities first and then disambiguated to match the mention to an entity definition. Consider the examples given below.

Ex1: President Obama met French President Macron at the White House today.

Ex2: The White House opposed the complete waiving of student debt.

Assume that the knowledge base contains an entity definition 'White House' with the definition being: "The physical location or building where the American president resides." In the first sentence, we have an entity mention of White House which matches with this definition. Hence, this entity mention can be disambiguated to the entity definition of the White House as a physical location. The second sentence also contains an entity mention 'White House'. However, this entity mention does not map to the physical location of the 'White House' but refers to the American president who occupies the White House. Hence, this entity mention cannot be disambiguated to the entity definition of White House as a physical location, even though the surface form of both the entity mentions match the entity definition.

We also have the case wherein the surface form of the entity mentions can be completely different from the entity definition due to inter-sentence references or context. For example, consider the sentence: *Potus is considering a ban on travel from high risk Covid countries*. This sentence contains an entity mention 'Potus', which actually refers to the president of the United States though there is no common surface form. Surface forms can completely differ for entity mentions from entity definitions due to inter-sentence references or external knowledge also. Consider the example given below.

Barack Obama is the first person of African-American origin to occupy the White House. The president was born in Hawaii.

In this case, the president is the entity mention which needs to be mapped to the entity definition 'Barack Obama', even though the entity mention surface form has no surface level similarity to the entity definition surface form. Such issues make entity disambiguation a challenging problem.

Early work in entity disambiguation typically used handcrafted features to map entity mentions to entity definitions. The task of entity disambiguation can be formally defined as follows: Given entity definitions from a knowledge base, and a text document annotated with entity mentions, the task is to map each entity mention to its corresponding entity definition including <*None*>. This can be treated as a multi-class classification problem modelled using supervised learning techniques, and can be addressed using classical machine learning techniques with handcrafted features. Of late, neural network based techniques have typically been used for entity disambiguation with recurrent neural networks (LSTMs) and transformer based encoders to capture the representation of the entity mention and entity definition. This learnt representation can be fed to a standard MLP classifier for the final classification.

Once entity mentions have been disambiguated to the corresponding definition, the idea is to learn the relation that exists between a pair of entities using the unstructured text. Consider the example we discussed earlier: 'The president was born in Hawaii'. Once the entity mention 'president' has been disambiguated to 'Barack Obama', as in the previous example, we are interested in identifying the relation present in the sentence containing the entity mention. In this case, it is straightforward as the predicate argument 'born' directly provides the relation. This allows us to extract the tuple <Barack Obama, 'Born', Hawaii> where Barack Obama and Hawaii are the entity definitions and the relation between these two entities is 'born'.

We know that entities have specific types. Relation classes, in fact, require that entities involved in the relation should be of certain types. In the above example, it is clear that the first entity should be a person. The second entity in this case is a geographical place. On the other hand, we can have a sentence 'The President was born in 1960', in which the second entity is of type <DATE>.

The fact that specific relation types require specific entity types is interesting, because it allows us to use the relation (if it is known) to disambiguate the entity mentioned. Hence there is considerable interaction between relation extraction (relation classification) and entity disambiguation. Therefore, instead of treating relation extraction and entity disambiguation as separate problems, a few works are now exploring these as a joint learning problem.

As we mentioned before, neural approaches are becoming popular in both entity disambiguation and relation extraction. Typical approaches include sentence representation using word embeddings, followed by a bidirectional LSTM layer (one or multiple layers stacked over each other), and a MLP classifier which uses the contextual representations from biLSTM layers to make the final prediction using softmax. Non-neural representations in earlier works used handcrafted features and also the dependency parse features between words in the sentence. Dependency parse related features enabled modelling of syntactic information. However, with the advent of neural approaches, explicit modelling of syntactic information was not considered necessary since the representations may implicitly learn syntactic features.

The issue with neural approaches was in handling long sentences, where the relation between two entity mentions was spread over a longer dependency path. This long dependency path became difficult for LSTMs to effectively capture in their representation, leading to poor performance in such sentences. Hence, there arose a need to reintroduce syntactic information in neural approaches for relation extraction.

The arrival of graph neural networks, namely, graph convolutional networks (GCNs) and graph attention networks (GATs) enabled leveraging them to add syntactic information in neural approaches for relation extraction. The key addition to a standard neural method for relation extraction is to add a GCN layer after the biLSTM layer. The GCN layer takes as inputs the biLSTM outputs and the dependency parse tree from an external dependency parser of the original input sentence. The outputs from the GCN layer are passed to the MLP classifier for the final classification. Each node in GCN is the biLSTM representation corresponding to each token, and the dependency parse tree is used to construct the graph between these nodes.

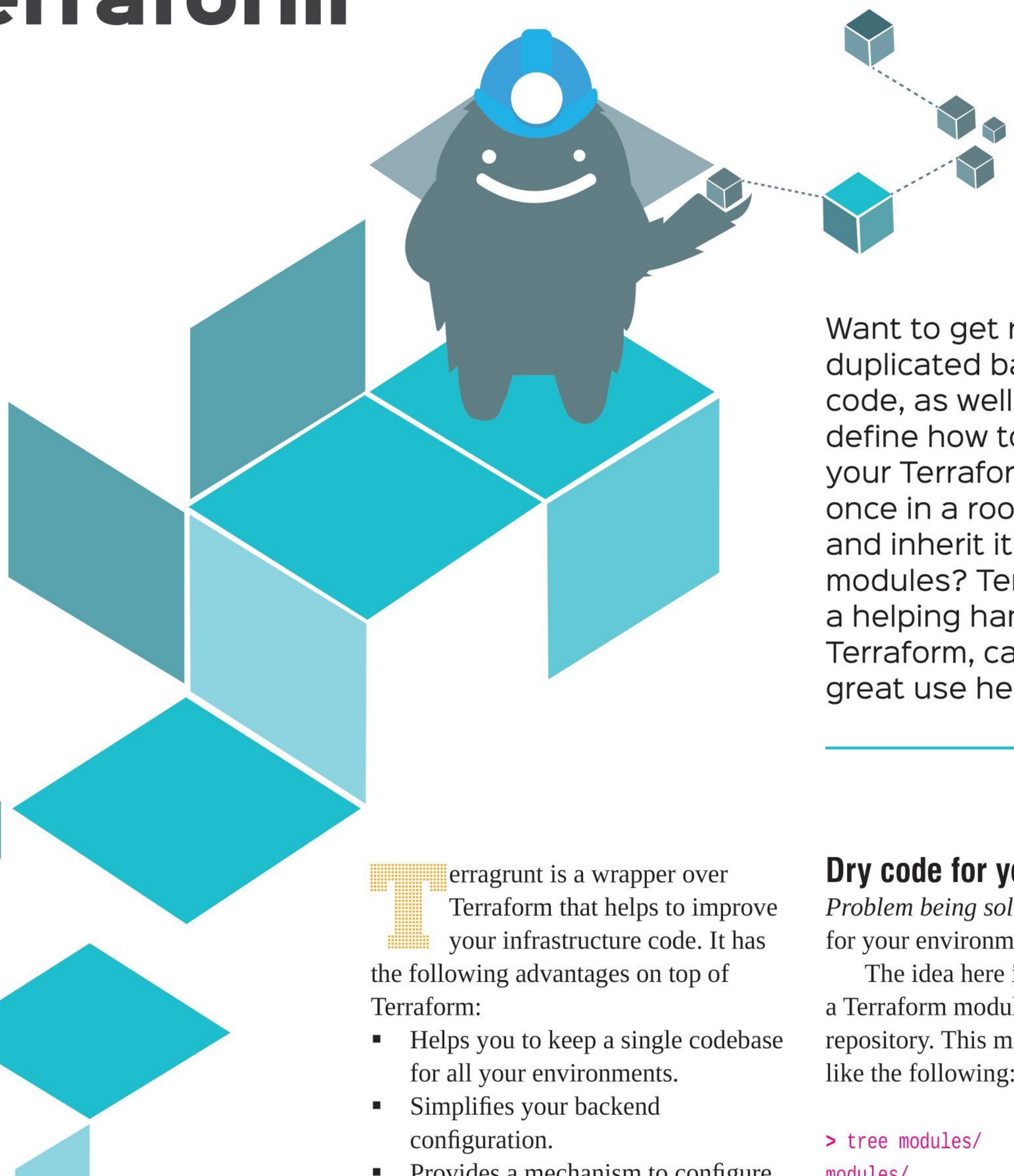
Note that two words that are wide apart in the linear sentence form are brought together by a direct dependency edge in their parse tree representation. Hence, adding the GCN layer enables the relation extraction model to handle the long-term dependencies easily. One thing to note is that standard GCNs represent undirected graphs, whereas graphs from dependency parse trees are directed. While it is possible to treat the dependency parse tree graph to be undirected, it results in loss of information. Hence, GCNs can be extended to handle directed edges by creating two edges with different labels (one between u, v and another between v, u). Also note that dependency parse trees contain labels for different edges. Hence, we need to use different parameter weights corresponding to different labels. However, this results in an increased number of parameters to be learnt. Often, systems employing GCN for representing syntactic representation avoid this parameter explosion by using different parameter matrices for different directions and not considering the label information. Only bias terms in the network representation are driven by the label information. This prevents over-parameterisation of the network. We will talk more about using GATs for relation extraction and entity disambiguation in our next column.

Feel free to reach out to me over LinkedIn/email if you need any help in your coding interview preparation. If you have any favourite programming questions/software topics that you would like to discuss on this forum, please send them to me, along with your solutions and feedback, at *sandyasm_AT_yahoo_DOT_com*. Wishing all our readers a very happy new year 2022! Stay healthy and stay safe. END

By: Sandya Mannarswamy

The author is an expert in natural language processing and is currently working as an independent researcher. Her interests include natural language processing, machine learning and AI.

Terragrunt:
The Orchestrator Tool for Terraform



Want to get rid of duplicated backend code, as well as to define how to manage your Terraform state once in a root directory and inherit it in all child modules? Terragrunt, a helping hand for Terraform, can be of great use here.

Terragrunt is a wrapper over Terraform that helps to improve your infrastructure code. It has the following advantages on top of Terraform:

- Helps you to keep a single codebase for all your environments.
- Simplifies your backend configuration.
- Provides a mechanism to configure your arguments for Terraform.
- Hooks for custom actions that may be needed before or after the execution of Terraform.

Terragrunt provides an orchestrator mechanism for managing multiple Terraform modules and provides the glue for driving them easily.

Dry code for your environments

Problem being solved: Single codebase for your environments (*dev, qa, prod*).

The idea here is that you define a Terraform module in a separate repository. This might look something like the following:

```
> tree modules/
modules/
├── app
│   ├── main.tf
│   └── variables.tf
└── vpc
    ├── main.tf
    └── variables.tf

2 directories, 4 files
```

The *main.tf* contains the relevant infrastructure code and *variables.tf* contains the input variables for the *app* and the *vpc* directories.

Now have a separate repository which will house the entire code for all your environments. It will look like this:

```
> tree infra
infra
├── dev
│   ├── app
│   │   └── terragrunt.hcl
│   └── vpc
│       └── terragrunt.hcl
└── prod
    ├── app
    │   └── terragrunt.hcl
    └── vpc
        └── terragrunt.hcl

6 directories, 4 files
```

Each Terragrunt module specifies a Terraform block, which specifies the location of the Terraform module. Along with that, you will also have to specify an input attribute to define the input values for the module. This is how a sample *terragrunt.hcl* will look for the dev/app environment:

```
terraform {
  source = "git::git@github.com:psibi/
infra-modules.git//app?ref=v0.1"
}

inputs = {
  instance_count = 2
  instance_type  = "m2.large"
}
```

If you want to go ahead and deploy just the *vpc* module for the dev environment, you can easily do that:

```
$ cd infra/dev/app/vpc
$ terragrunt plan
$ terragrunt apply
```

Simplify backend configuration

Problem being solved: Terraform doesn't allow variables or expressions in its backend configuration. Terragrunt solves this without any hardcoding of values.

Taking the above infrastructure example, for dev environment *vpc* we want the following kind of state:

```
terraform {
  backend "s3" {
    bucket         = "my-infra-state"
    key            = "dev/vpc/
terraform.tfstate"
    region         = "us-east-1"
    encrypt        = true
    dynamodb_table = "infra-lock-table"
  }
}
```

Terragrunt allows you to solve this by putting a root *terragrunt.hcl* in the repository:

```
> tree infra
infra
├── dev
│   ├── app
│   │   └── terragrunt.hcl
│   └── vpc
│       └── terragrunt.hcl
├── prod
│   ├── app
│   │   └── terragrunt.hcl
│   └── vpc
│       └── terragrunt.hcl
└── terragrunt.hcl

6 directories, 5 files
```

Now you can define your backend configuration once in the root *terragrunt.hcl* file:

```
remote_state {
  backend = "s3"
  generate = {
    path      = "backend.tf"
    if_exists = "overwrite_terragrunt"
  }
  config = {
    bucket = "my-infra-state"

    key = "${path_relative_to_
include()}/terraform.tfstate"
    region         = "us-east-1"
    encrypt        = true
    dynamodb_table = "infra-lock-table"
  }
}
```

All the heavy work here is done by the *remote_state* block and Terragrunt. The function *path_relative_to_include* block does the main work by giving unique backend keys for each of the folders. Inside *prod/app* it would have returned the value prod/app. This will make sure that you have unique key names in your bucket. For the above configuration to work, you also have to update the child *terragrunt.hcl* files with the following:

```
include "root" {
  path = find_in_parent_folders()
}
```

When to use Terragrunt

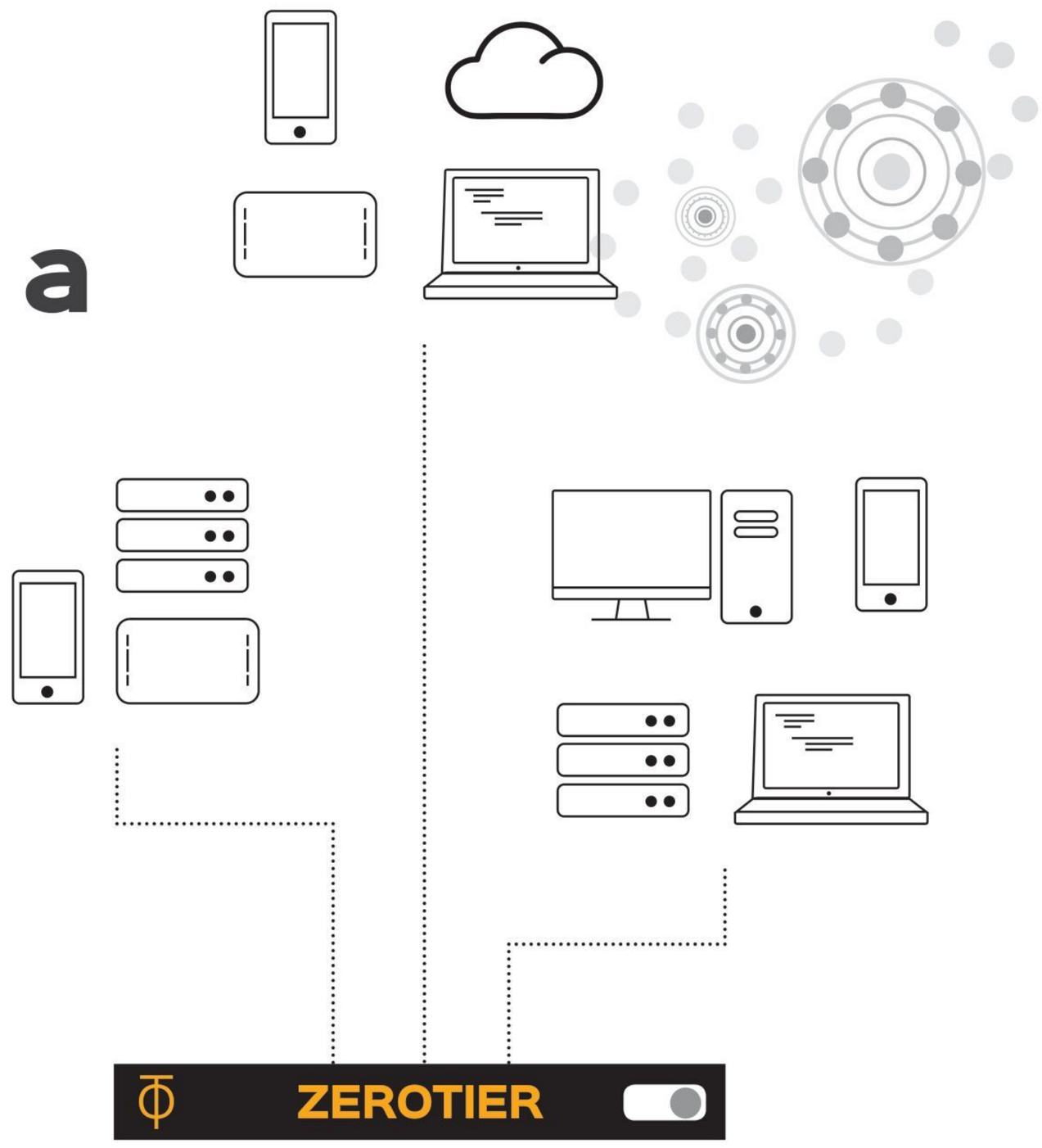
While Terragrunt is nice and useful, I recommend using Terraform if you don't have various environments to manage or if you are just starting out in your Infrastructure-as-a-Code journey.

Terragrunt solves real painpoints by simplifying backend configuration for Terraform when you have multiple environments with various modules to manage. I hope this post will make you think of and consider using Terragrunt when you write your infrastructure code next! END

By: Sibi Prabakaran

The author is a senior engineer working at FPComplete.

ZeroTier: The Internet as a Global Switch



ZeroTier describes itself as a global Ethernet switch to connect any number of devices. The subscribers connect to this imaginary switch and then communicate as if they were in the same local network. Let's look into how this actually works.

ZeroTier's Ethernet switch is figurative and symbolic; subscribers actually connect via a virtual private network (VPN) to a predefined VPN server operated by ZeroTier. The VPN server connects clients' devices so that they can talk directly to each other. The result is an encrypted peer-to-peer network.

The transport network is responsible for securing participants' communication and authentication. This network also achieves another important task by finding the best path between clients. Above this network layer, ZeroTier emulates the well-proven Ethernet to provide applications with a known protocol. For the participants, the VPN appears as an Ethernet switch. The technology behind this design is similar to IPsec and VXLAN.

The data packets between ZeroTier clients are encrypted and only understood by other clients, but not by the ZeroTier servers. The used algorithm originates from the toolbox of elliptic curves and offers a key length of 256 bits. The client software automatically handles the key exchange and certificate management.

In Figure 1, the clients use the ZeroTier cloud as a fully meshed backbone network that independently finds the shortest connections between all sites.

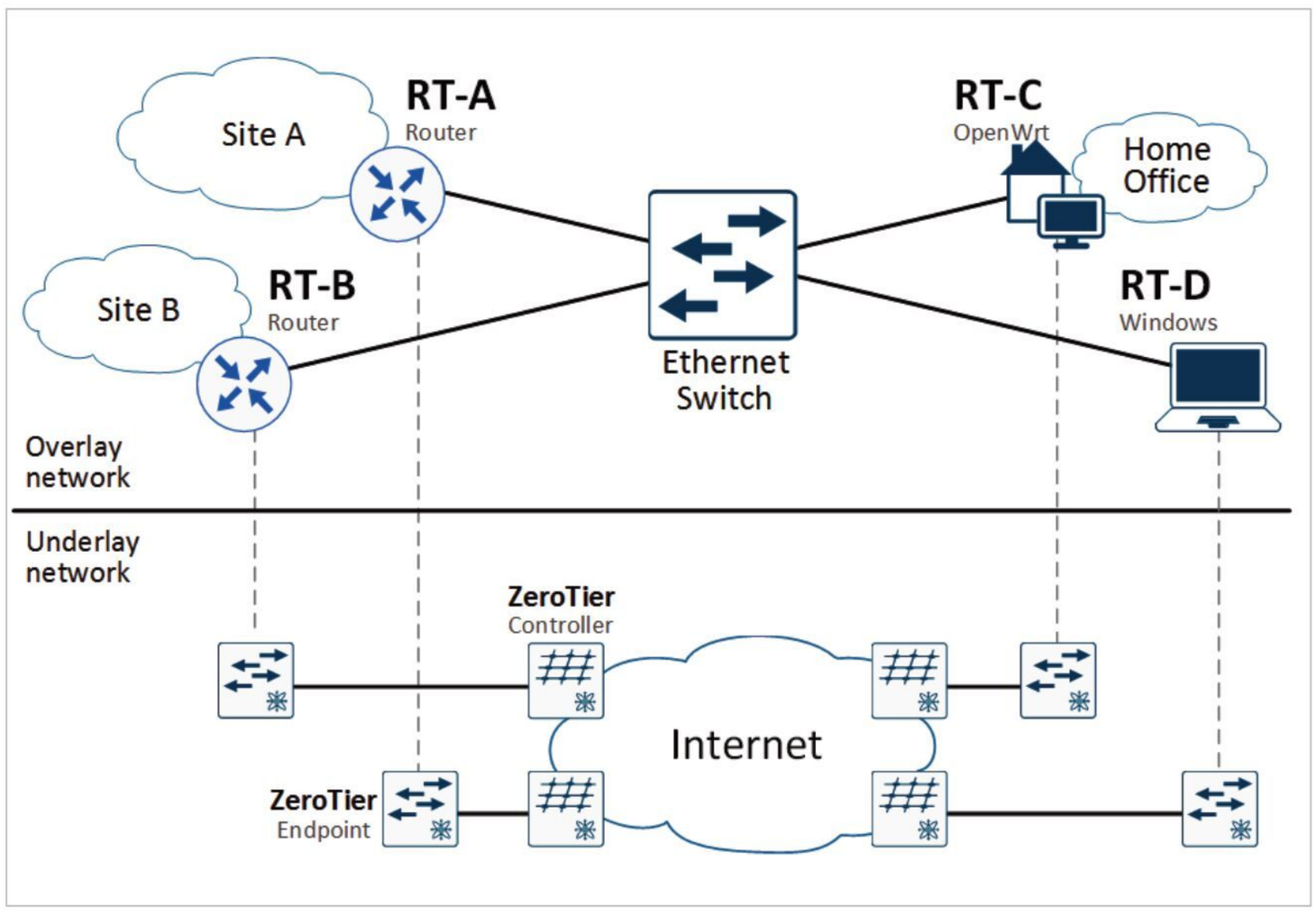


Figure 1: Various clients connect through the ZeroTier switch

Access

The computer uses the client software ‘ZeroTier One’ to access the ZeroTier network, which is offered as source code and as a precompiled binary for many operating systems and smartphones. On a laptop, the ZeroTier switch is another network adapter with IP address, routes, and firewall rules. It can be used to connect remote sites, dial in clients, or create out-of-band management access.

To begin, the Web service ‘ZeroTier Central’ expects a registered user. The registration process creates a new Ethernet switch. In return, ZeroTier provides a network identifier that the clients use to access the network. This network identifier distinguishes the clients from each other and prevents client A from reaching the devices in client B’s network. Keep in mind that ZeroTier is a free commercial service provider for up to 50 devices.You can use ZeroTier’s Web portal to register a new account. Afterwards, create a network for your devices by clicking on the button ‘Create A Network’ in the Web UI of ‘ZeroTier Central’. The network immediately appears in the right-hand area with a randomly generated ID and an exemplary name. In this example, the new network has chosen the ID 1d7193940486c4c6, which is required later for the ZeroTier One client setup. Click on the network ID to configure the properties. The preset values are suitable for the initial setup.

Installation

On the client-side, the device needs the software ‘ZeroTier One’, which is available for all major operating systems. The installer script for Linux recognises the distribution and uses the appropriate package from the repository:

```
curl -s https://install.zerotier.com
| bash
```

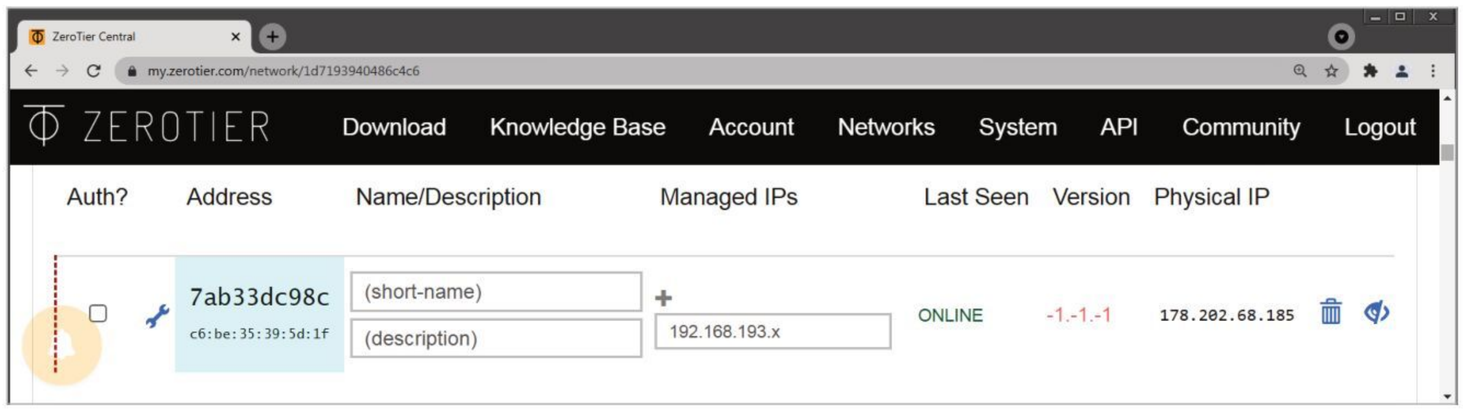


Figure 2: The new subscriber appears as a non-authorised client

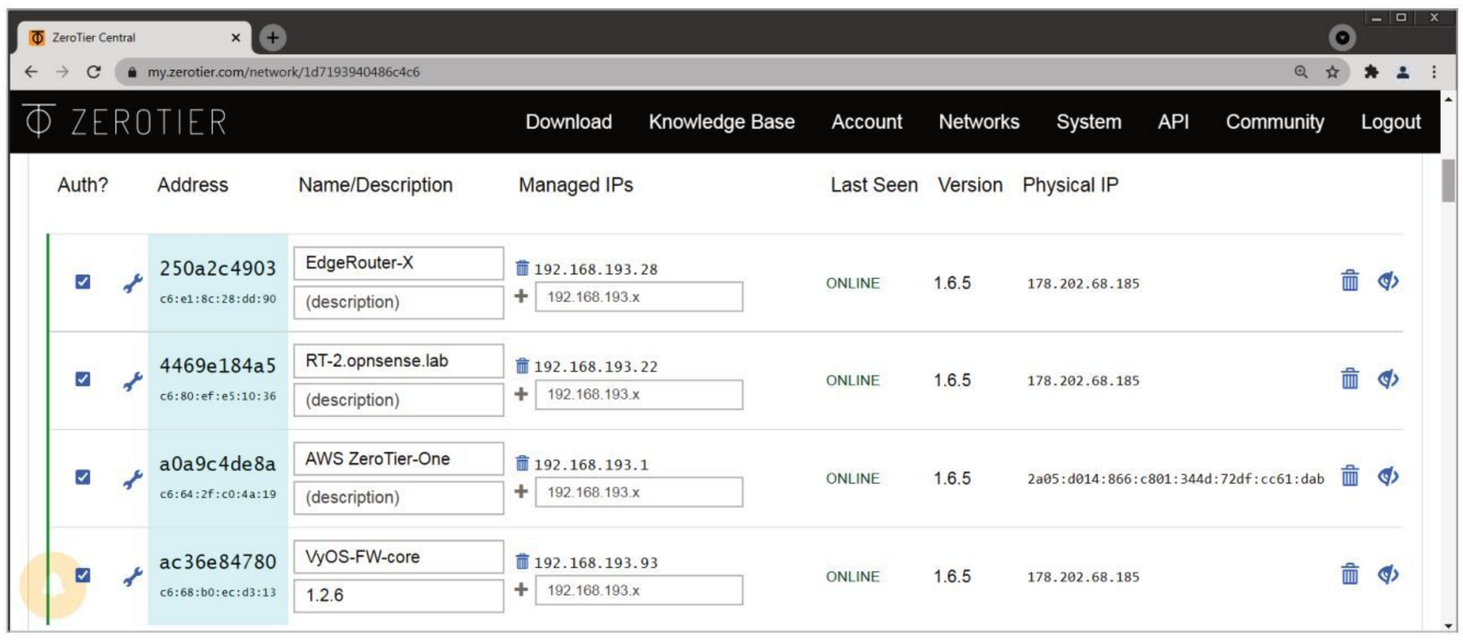


Figure 3: Multiple clients have registered with ZeroTier Central

Use the command line to set up a connection between your computer and ZeroTier. The command requires the network ID from the previous section and then connects to the ZeroTier server.

```
zerotier-cli join 1d7193940486c4c6
```

The Web view of ZeroTier and Figure 2 show the newly connected client. Since the default settings define the network as ‘private’, the client can only use the ZeroTier network once the admin has confirmed this in the ‘Auth’ column. It is also helpful to label the new device to easily distinguish it from future clients.

Note: The ZeroTier client assigns the interface a cryptic name that begins with the letters ‘zt’. This name may differ from your own experiments in the following command outputs.

As soon as all devices are connected, the list in ZeroTier extends and becomes similar to Figure 3. For validation purposes, the clients can ping each other using their ZeroTier IPv4 address from the assigned range (here: 192.168.193.0/24). The setup under Windows, macOS, and on smartphones is similar, except for a graphical interface.

ZeroTier as a switch

ZeroTier compares itself to a global Ethernet switch, allowing clients to use the ZeroTier connection as a network bridge. The result truly feels like an Ethernet connection to the subscribers.

By default, clients do *not* operate as a network bridge. To enable bridge mode, go to the ZeroTier Web UI and enable ‘Allow Ethernet Bridging’ for each device individually (Figure 4).

On the client-side, Linux requires a network bridge that joins the interfaces eth0 and zt7nnn53ro. The result is a genuine Ethernet connection; the connected devices share a broadcast domain and can use the same IP network. There is also no need to manage and distribute routes.

Firewall

Linux and Windows come with a robust packet filter that can block unwanted IP connections. The use of this local firewall is made possible

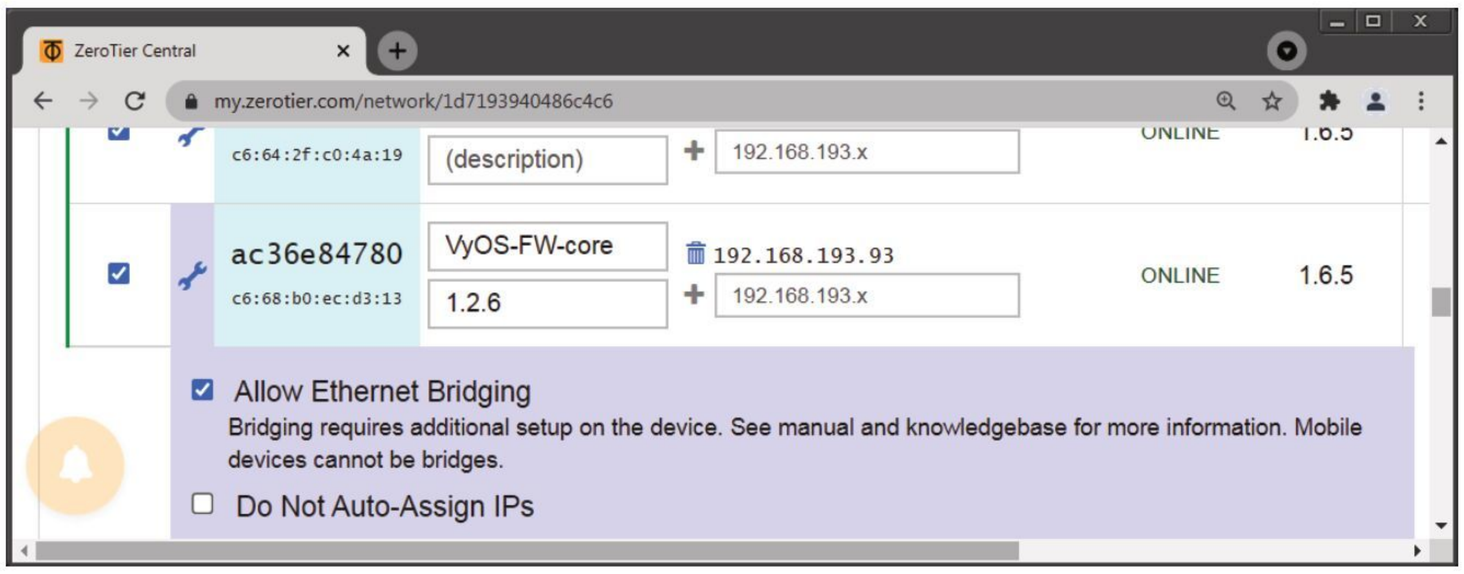


Figure 4: ZeroTier allows 'Ethernet bridging' per device

because ZeroTier is treated as a regular interface and can belong to a firewall zone.

ZeroTier Central comes with 'Flow Rules', which is an effective way to control the data streams from a central point. The configuration is less convenient; the firewall rules are created as text, similar to iptables rules. The resulting security policy applies to all ZeroTier clients connected to this network ID. As soon as the rule set is ready, the clients fetch the new policy and apply it locally.

The use of the firewall is recommended if incoming traffic originates from an untrusted location or if the ZeroTier switch is classified as insecure. In this case, the ZeroTier interface will receive a firewall policy that uses restrictive rules to block anything that is not explicitly allowed. The firewall policy can filter at both the IP layer and the Ethernet layer.

Self-hosting ZeroTier

The makers of ZeroTier maintain several root servers for all subscribers so that the client software can get in touch with the network and reach other nodes. It is possible to run one (better: two) root server and a network controller to independently build a ZeroTier cloud without outside dependencies. The required software is already installed because the 'ZeroTier One' client also functions as a network controller or root server.

Network controller

The first step to independence is to have your own network controller. ZeroTier uses the network controller as a configuration manager and certification authority. The configuration of the controller uses ZeroTier's API, which is well documented but inconvenient for daily use. Those who do not like to set up their new infrastructure with *curl*, *wget*, or *httpie* can use the Web interface from the third-party provider 'Key Networks'. The installation of the ZeroTier software and the Web UI is described in Listing 1 for a server running Debian Linux.

Listing 1: This script turns a Debian Linux host into a ZeroTier controller

```
wget http://download.zerotier.com/
debian/buster/pool/main/z/zerotier-one/
zerotier-one_1.8.1_amd64.deb
dpkg --install zerotier-one_1.8.1_
amd64.deb
systemctl restart zerotier-one

wget https://s3-us-west-1.amazonaws.
com/key-networks/deb/ztncui/1/x86_64/
ztncui_0.7.1_amd64.deb
dpkg --install ./ztncui_0.7.1_amd64.deb
echo 'HTTPS_PORT=8443' > /opt/key-
networks/ztncui/.env
systemctl restart ztncui
```

After the installation is complete, the Web interface is available at the URL *https://ipaddress:8443/*. The first login uses the credential admin/password. Subsequently, the Web UI forces a password change to boost security.

The configuration is very similar to the procedure from the previous section, although the Web UI of 'Key Networks' offers a smaller functionality than ZeroTier Central. The setup starts with the button 'Add network', which immediately creates a new network and presents the network ID. A client can directly join this network without knowing the network controller. The required command is no different from a regular 'join':

```
zerotier-cli join a0a9c4de8a520e4d
```

In the network controller's Web interface, the new members need authorisation and an IP address. In

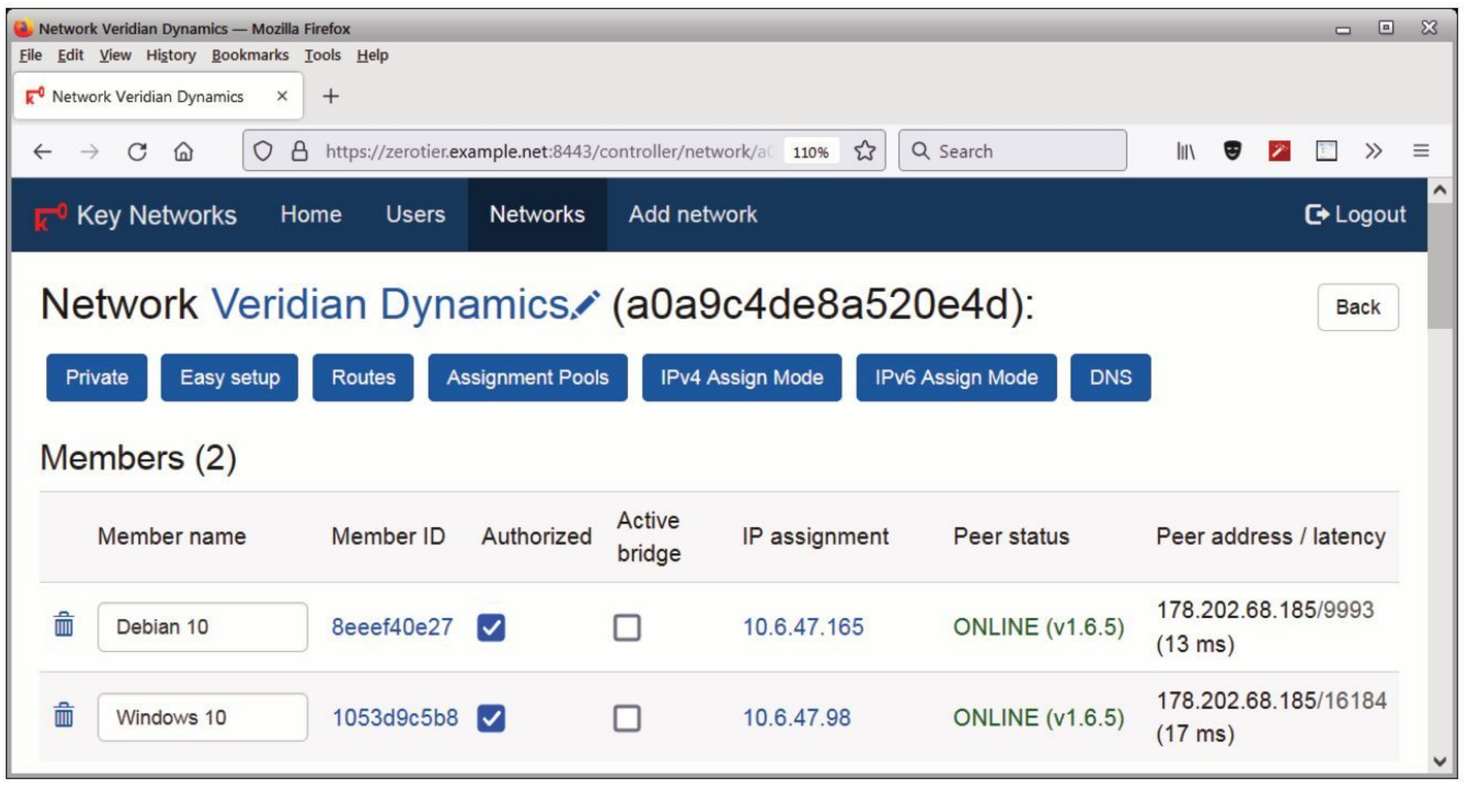


Figure 5: The clients have registered with the self-hosted network controller

Figure 5, the participants have registered with the controller and received their IP addresses. In the background, the ZeroTier clients communicate with the ZeroTier root servers to find the new controller. The next step to independence is to have your own root server.

Root server

ZeroTier implements the functionality of the root server again in its 'ZeroTier One' application. The documentation refers to a self-hosted root server as a 'moon'. The provider recommends using two moons that have a stable network connection and static IP addresses. The hardware requirement is minimal, so any virtual machine at a cloud provider can become a root server. Furthermore, the provider recommends that the root server does not simultaneously work as a network controller.

In Listing 2, a Debian server becomes the moon of the ZeroTier universe. Line 2 creates a world definition as well as crypto material. Line 4 lists the new root server's IP address (here: 18.192.207.139) as the 'stable endpoint'. If a second moon is added, it needs another entry under 'roots'.

Listing 2: The Debian machine becomes a root server

```
cd /var/lib/zerotier-one/
zerotier-idtool initmoon identity.public >> moon.json
sed -i -e '
s!stableEndpoints":.*!stableEndpoints": [ "18.192.207.139/9993" ]!
' moon.json
zerotier-idtool genmoon moon.json
mkdir -p moons.d/
mv 000000a0a9c4de8a.moon moons.d/
systemctl restart zerotier-one
```

The resulting moon file is signed in line 6 and stored in the subdirectory *moons.d*. The file name is inspired by the world identifier, which is created in line 2. The clients will need this identifier later to find the moon. The root server is ready to use only after restarting ZeroTier One.

Finally, it is time to point the ZeroTier clients to the new universe. In the technical language of ZeroTier, the client 'orbits' the moon. Specify the world ID to guide the client to the new moon from Listing 2, whereby the leading zeros of the ID may be omitted:

```
root@client:~# zerotier-cli orbit
a0a9c4de8a a0a9c4de8a
200 orbit OK
```

The command 'zerotier-cli peers' shows if the client and moon meet. Executed on the moon, the command output should include the IP addresses of the other clients and show the role LEAF. Executed on the client, the list of planets should contain the self-hosted moon in the role MOON (Listing 3).

Listing 3: The client has made contact with the new moon

```
root@CentOS8 ~> zerotier-cli peers
200 peers
<ztaddr>   <ver>  <role> <lat> <link>
<lastTX> <lastRX> <path>
1d71939404 1.8.0  LEAF    -1   DIRECT
17771 17625 34.70.132.9/28575
61d294b9cb -      PLANET  181  DIRECT
2658  2476  50.7.73.34/9993
62f865ae71 -      PLANET  245  DIRECT
2658  2415  50.7.252.138/9993
778cde7190 -      PLANET  137  DIRECT
7671  2510  103.195.103.66/9993
992fcf1db7 -      PLANET  30   DIRECT
12680 2617  195.181.173.159/9993
a0a9c4de8a 1.8.1  MOON    30   DIRECT
2623  2623  18.192.207.139/9993
```

The ZeroTier clients and the new moon still have connections to other nodes in the ZeroTier universe. As soon as these are no longer accessible, the remaining infrastructure continues to operate autonomously and is thus independent of external services or an Internet connection.

Technical background

ZeroTier hides its complexity behind a fancy Web interface and the 'ZeroTier One' software. In the background, a distributed network hypervisor operates over an encrypted peer-to-peer network. ZeroTier divides its complexity into an overlay network, which emulates an Ethernet switch for its subscribers; and an underlay network, which uses the shortest path to deliver the packets to the end devices (Figure 1).

The transport network consists of several root servers, which link the ZeroTier clients to each other and provide hints for short end-to-end connections. If the clients cannot find a suitable connection to each other, the root servers forward the packets. The root servers are operated by ZeroTier and thus make up the backbone network.

Above the transport underlay network, the Ethernet virtualisation layer serves as the overlay network. It presents the ZeroTier network as an Ethernet adapter to the operating system and the applications. For this to be successful, the overlay network must handle the typical tasks and protocols of Ethernet — ARP, NDP, broadcast, and multicast.

The network controller is responsible for the configuration and also acts as a certification authority, issuing certificates for the endpoints. The ZeroTier One software can be used as a client, network controller, or root server.

Routing

ZeroTier can not only connect individual clients, but also entire networks. The ZeroTier software

then runs on a router that announces its connected IP networks to the other ZeroTier routers. The outcome resembles a multipoint VPN that connects all locations and is managed from a central location. The advantage of this solution is that it eliminates the manual configuration that is typical for OpenVPN and IPsec.

The best operating system for the new ZeroTier router is a Linux/UNIX distribution that focuses on routing and firewalls, such as OpenWrt, VyOS, and OPNsense. The ZeroTier software is installed via a package manager and requires the well-known network ID from the previous sections. After authorisation in ZeroTier Central, the new routers are regular members of the ZeroTier group.

The routers now recognise each other, but they do not yet know all the attached local networks. There are several possibilities for exchanging routing information:

- The routers learn the path information to the other locations from the ZeroTier server. For this to work, enter the IP network in the field 'Managed Routes' of ZeroTier's Web interface (Figure 6). The ZeroTier clients will learn from this central routing table upon connection. This method is suitable for all environments as long as all routers have a connection to ZeroTier. In the free version of ZeroTier, the maximum number of routes is 32.
- Alternatively, the routers use a dynamic routing protocol, such as OSPF. The routers establish neighbourships through the ZeroTier network and use them to exchange IP information. This method is suitable for large environments or if some routers do not have a connection to ZeroTier.

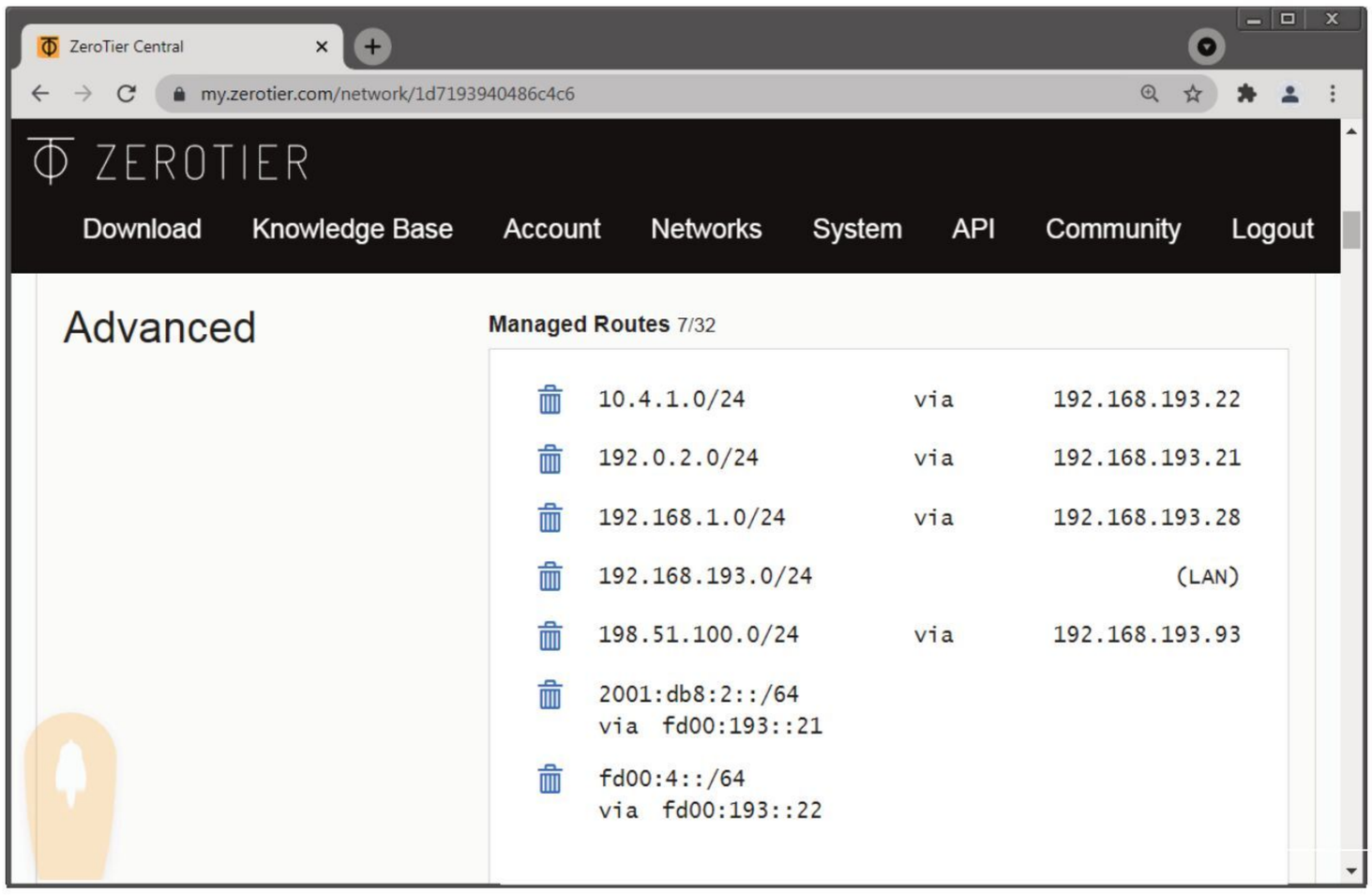


Figure 6: ZeroTier manages and distributes IP routes to its clients

Outlook

The design of ZeroTier's VPN is similar to a software-defined wide-area network (SD-WAN). In fact, ZeroTier with its central routing table, flow rules, and multipath has many similarities to SD-WAN. However, ZeroTier's VPN is missing a zero-touch deployment of routers and granular control over which traffic streams should flow over which VPN connection.

Name resolution of ZeroTier devices was a weak point for a long time, so users mostly had to work with static IP addresses. As a workaround, there are community projects, like ZeroDNS, and finally, ZeroTier provided the official DNS service 'ZeroNSD' — but only in the beta stage.

Those who find the 'Key Networks' Web interface lacking functionality, can switch to the full-blown software ZeroUI. The software has a large number of dependencies and only runs in a Docker container. After a rather complex installation, a full-fledged Web UI is available that implements virtually all of ZeroTier's functions.

Summary

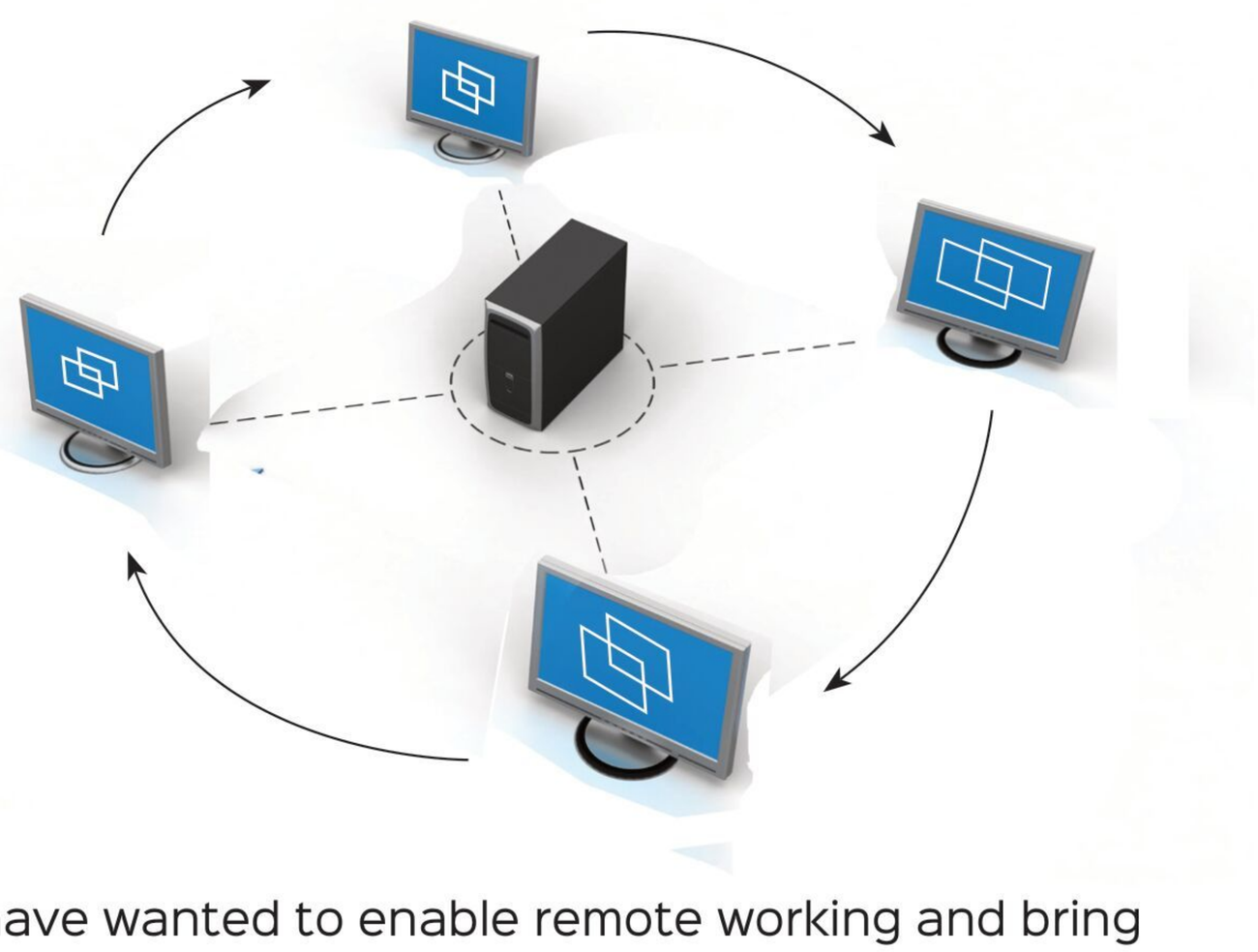
ZeroTier is a VPN solution that is easy to configure and is usable on all major operating systems and smartphones. The 'ZeroTier One' client automatically negotiates the tunnels, algorithms, and keys. To access the global network, the client only needs a network ID, which the administrator generates after registering in ZeroTier's Web portal. As soon as all the clients are registered, they can reach each other and find the shortest paths between them.

The initial contact of the ZeroTier client is always against the provider's root servers. An Internet connection is mandatory, even if the devices are located in the same network segment. This dependency is compensated by simple configuration, great flexibility, and high security. END

By: Markus Stubbig

The author is a networking engineer who has worked in the IT industry for 16 years. He focuses on design and implementation of campus networks around the world.

Remote Working and BYOD: Can Virtual Desktop Infrastructure Secure Your Enterprise?



CIOs have wanted to enable remote working and bring your own device (BYOD) policies since long, but never took the tasks up as a top priority. Covid-19 changed that and compelled every enterprise to explore both. Network and security teams, which used to raise objections earlier, have realised that remote working and BYOD are inevitable, and are redesigning enterprise ecosystems to enable them. In this article, we will explore virtual desktop infrastructure (VDI) as a solution for securing enterprises that have implemented remote working and BYOD policies.

The pandemic has enhanced the demand for work from remote environments. As a result, OEMs, infra and cloud providers, application vendors, researchers, analysts, and service integrators have all come together to explore solutions and develop the products to meet this demand.

Lockdowns imposed due to the pandemic made enterprises quickly explore various options to enable employees to contribute to enterprise productivity while working from home. Some companies purchased a large number of new laptops and couriered these to the homes of their employees. They installed VPN agents and set up VPN systems on a war footing. Though these offered a quick fix solution, they had serious limitations. With VPNs, the enterprise ecosystem is exposed to the outside world. Compromised end user devices and networks created many vulnerabilities, and opened the doors for hackers. Laptop procurement, too, had supply chain issues. Employees not satisfied with the laptops provided by the company requested that they be allowed enterprise application access from their personal device. CIO teams have long been mulling over the bring your own device (BYOD) policy for their organisations, too.

VDI as a solution

Remote work and BYOD require a redesign of the end-to-end enterprise ecosystem. Virtual desktop infrastructure (VDI) is meant for just this. VDI technology has a long history, and has matured enough to meet any enterprise remote working needs. It is built from scratch with a focus on security, the user experience and agile operations.

Work from anywhere, any time, using any device requires an elevated security posture and can't depend on existing castle models. The zero trust security concept that has lately become popular in the security community rightly reflects and fits well into the remote work and BYOD scenarios.

Security vulnerabilities with remote work and BYOD

Figure 1 shows the threat landscape in remote work and BYOD scenarios. End user devices are susceptible to viruses and malware. Unless the user diligently takes care to keep the OS and applications updated with the latest security patches and leverages industry standard anti-virus software, devices are vulnerable and can easily get compromised. When these compromised devices are used to access enterprise resources, there is every chance the malware finds its way into the enterprise network. Once it gets

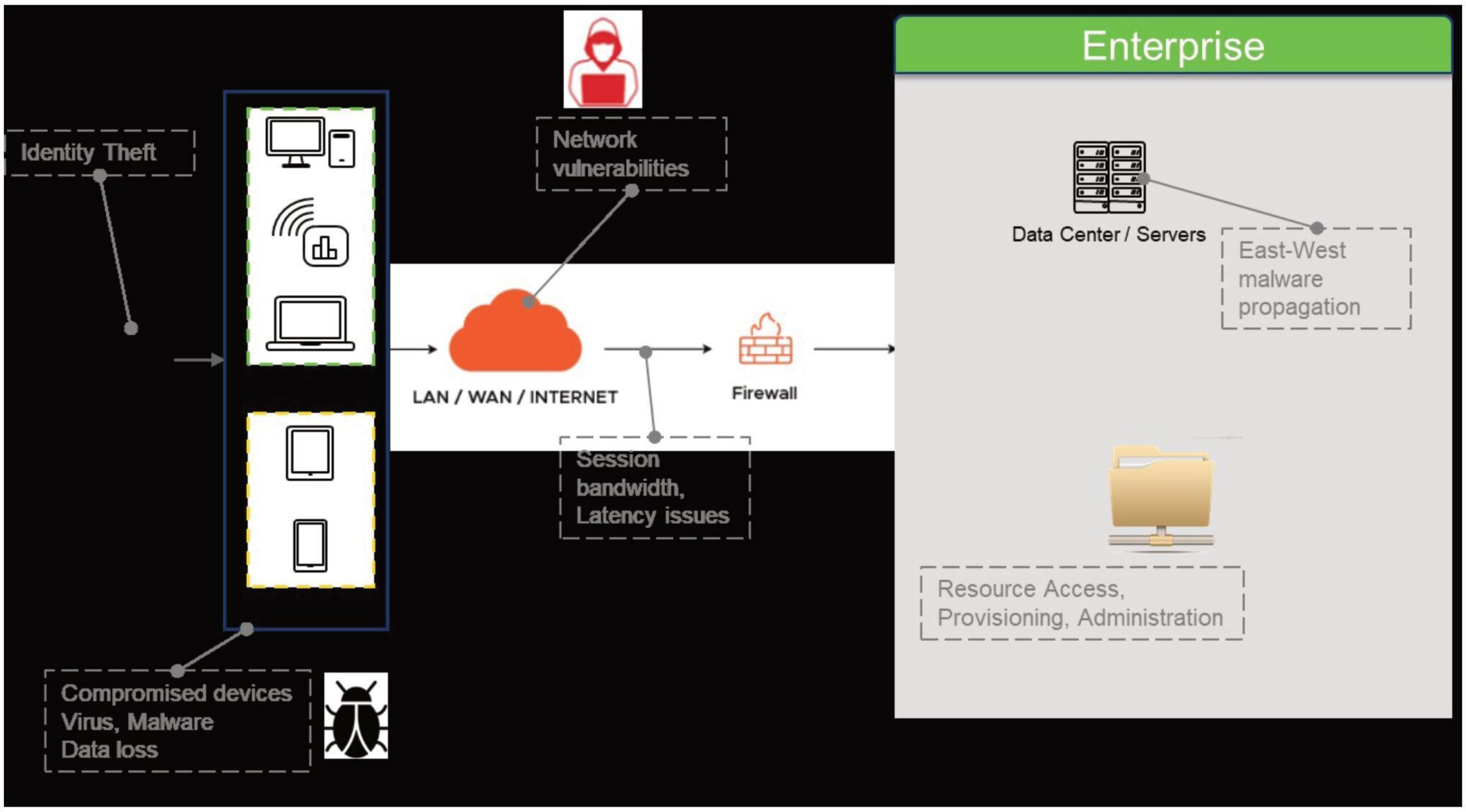


Figure 1: Threats and issues with remote work and BYOD

there, it propagates laterally in an east west direction to take control over more resources. Data gets transferred to the end point and can get into the wrong hands. User sessions over the network are prone to hacking.

How VDI handles security vulnerabilities

VDI is built from the ground up to address the various threats mentioned just now. Figure 2 summarises the controls imposed at each stage by VDI.

VDI leverages proprietary protocols for session streaming to the end point. Popular protocols in use are ICA from Citrix, and PCoIP and Blast from VMware. User sessions are encrypted end-to-end so that hackers cannot tamper with them during transmission.

Multiple levels of authentication are enforced for better security and to protect identity hijacks. Tokens, biometrics, and certificates over another communication channel are leveraged as second factor authentication to ensure only the right user gets access.

No data gets transferred to or from the end user device. Posture checks and conditional access allow deployment of security policies. VDI vendors have amplified security controls with anti-keyloggers and screen capture disablers to protect enterprise data and applications.

Even though network segmentation is not an integral part of VDI, it complements the VDI security posture in data centres to control east west traffic. This is important, as the adversary who somehow manages to pass through enterprise doors tries to move laterally to gain control over more and more devices and services in the enterprise. Network segmentation imposes security checks at each component, which stops unwanted east west traffic.

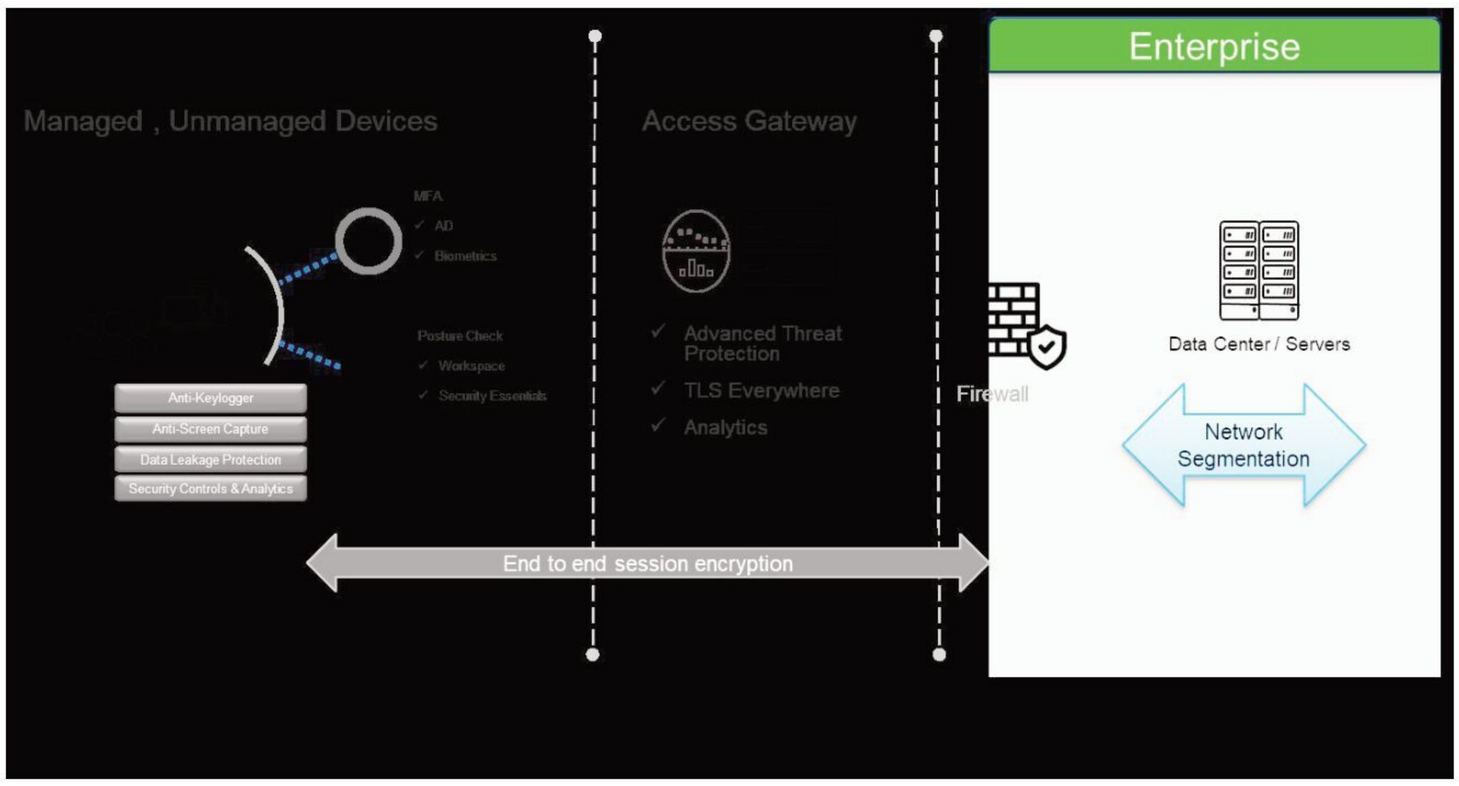


Figure 2: Security measures in a VDI landscape

Conditional access

Conditional access is a very important element in the VDI security posture. It includes a contextual policy engine that runs posture checks on end points and

Threats and risks	Levers
Data leakage	Policies to control client drive mapping, restrict downloads, disable clipboard mapping and enable watermarks
Threats from malware	Enable anti-keylogger and anti-screen capture features of App Protect
Social engineering and identity theft	Multi-factor authentication using a combination of biometric checks and active directory credentials; support for SAML (security assertion markup language) based identity providers. Posture checks and App Protect provide added security
Attack surface	Single URL endpoint, TLS (transport layer security) everywhere, and advanced threat protection against attacks such as network and app DDoS
Lateral propagation	Network segmentation enables security policies, isolation and traffic control

Table 1

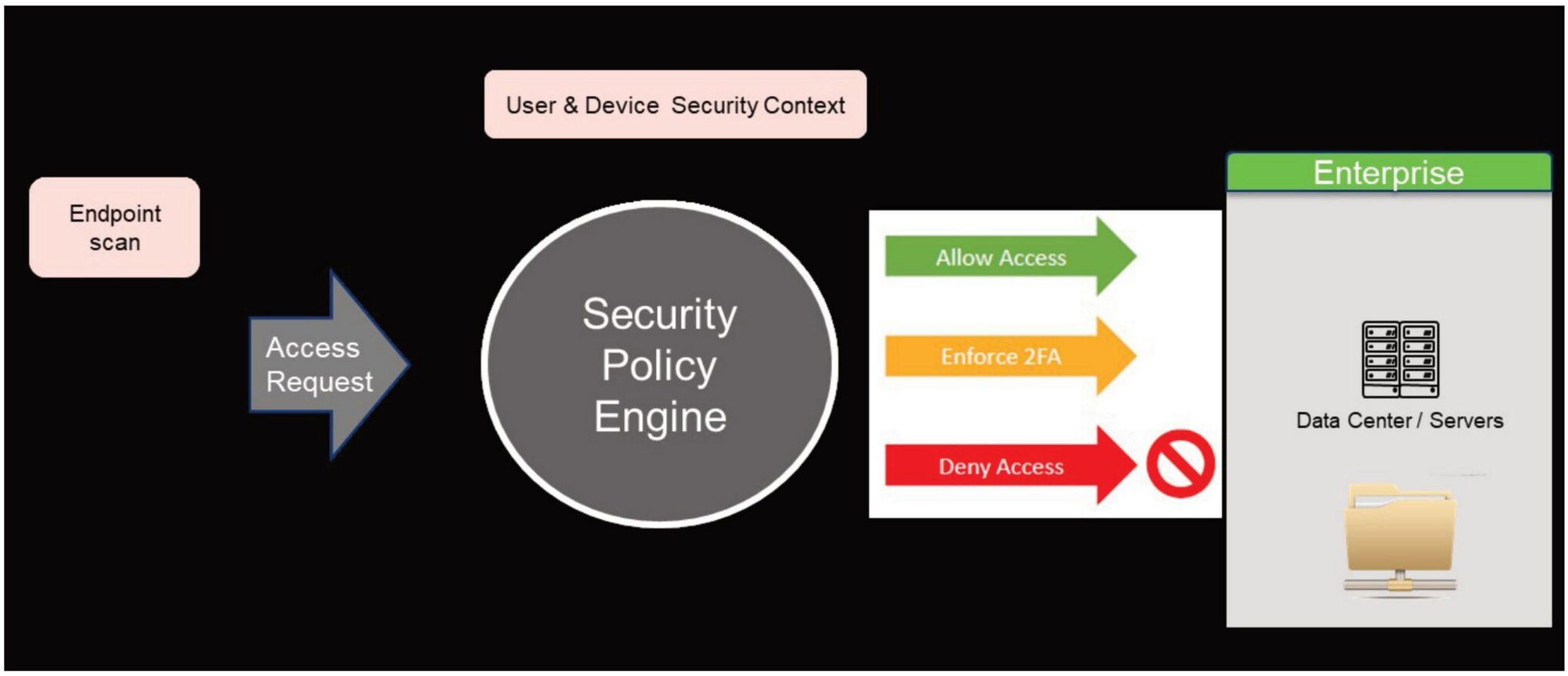


Figure 3: Conditional access

collects details to decide access conditions. Based on enterprise policies, user access is either allowed, denied or additional authentication checks are enforced.

In addition to the above security controls, VDI also supports forensic analysis with watermarks, audit reports and analytics to provide a better security posture.

Table 1 summarises the various levers built into VDI to seal security vulnerabilities.

Even though VDI is built with tight security levers, it is still not 100 per cent fool-proof. Human mistakes coupled with hacker intelligence can still put VDI security at risk. However, VDI does raise the security bar to a very high level, which makes it very hard to compromise it. END

References

[1] Gartner; *https://www.gartner.com/smarterwithgartner/new-to-zero-trust-security-start-here*
[2] Citrix; *https://docs.citrix.com/en-us/tech-zone/learn/tech-briefs/zero-trust.html*
[3] VMWare; *https://www.vmware.com/in/remote-work-solutions.html*

By: K. Narasimha Sekhar and Radhakrishna Singuru

K. Narasimha Sekhar has 27 years of experience in product development for infrastructure management and cloud, as well as telecommunications, manufacturing and technical delivery. He is the principal architect for virtual desktops and end user services at Wipro Technologies and DMTS - senior member.

Radhakrishna Singuru has more than 25 years of industrial experience in product and system software development spanning multiple industry domains. He is chief architect, cloud platform products and security engineering practice, ER&D at Wipro Technologies and DMTS - senior member.

Disclaimer

The views expressed in this article are that of the authors and Wipro does not subscribe to their substance, veracity or truthfulness.

EFY GROUP
YOURS SINCE 1969

Thank You IIT Madras

It was in your hostel room that the idea was conceived

The first issue in Jan '69

The Jan '21 Issue

Celebrating 53 years in TECH media

Magazines
- Electronics For You
- Electronics Bazaar
- Open Source For You

Portals
- www.electronicsforu.com
- www.opensourceforu.com
- www.eleb2b.com
- www.electronicsb2b.com

Events
- India Electronics Week
  - IOTshow.in
  - EFY Expo
  - Test & Measurement India
- Open Source India
- Techworldcongress.com

EFY Enterprises Pvt Ltd
D-87/1, Okhla Industrial Area, Phase-1, New Delhi-110020

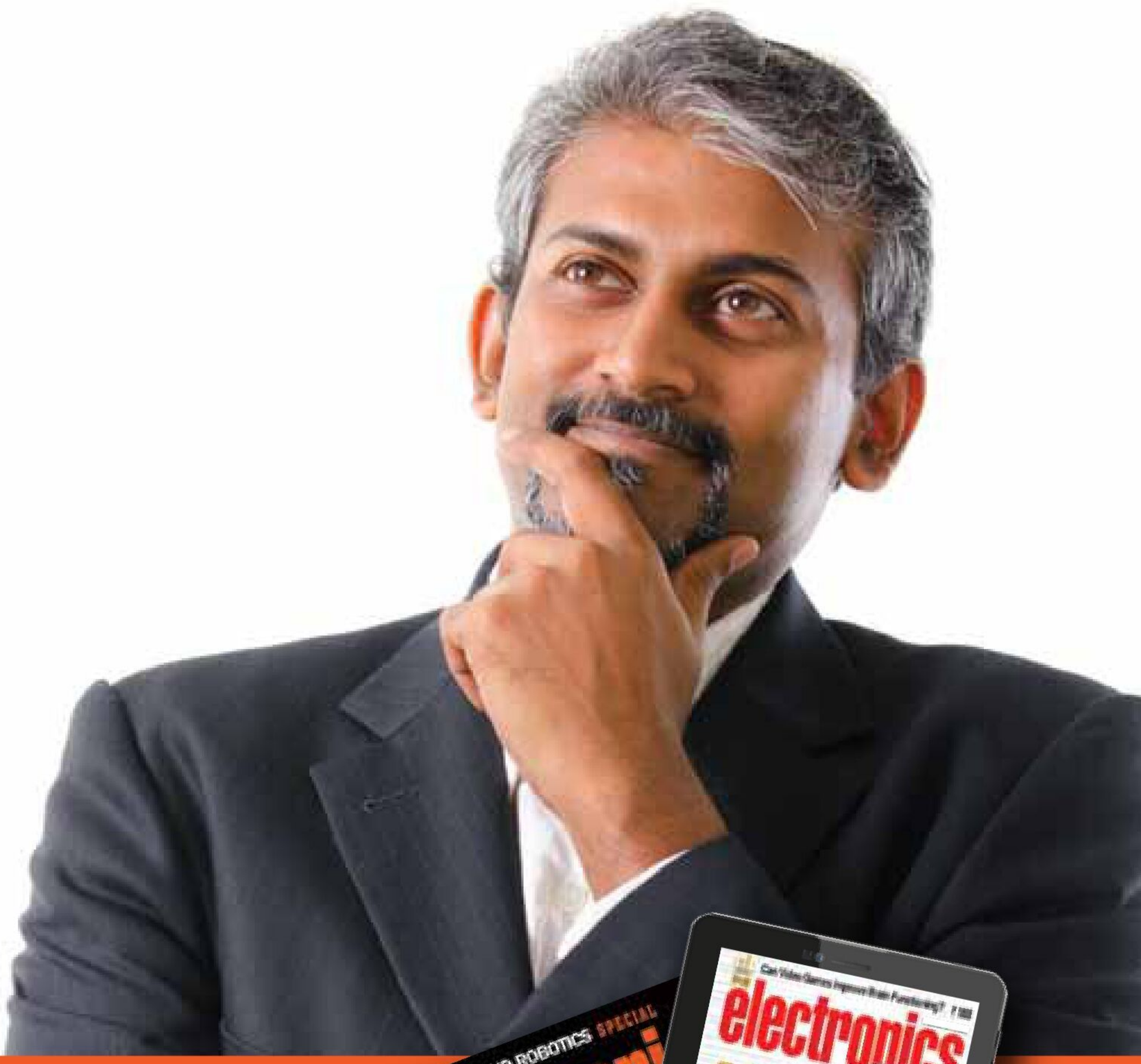
• Phone: +91-98111-55335
• Email: info@efy.in • Website: www.efy.in

R N I No. DELENG/2012/49440, Mailed on 27/28th of Advance month
Published on 27th December 2021
Delhi Postal Regd. No. DL(S)-01/3443/2022-24
Licenced to Post without Pre-Payment Licence No U(S)-56/2019-21
Posted at LPC Delhi RMS Delhi 110006

India's No.1 Electronics Magazine
RANKING AS PER ELCINA-IMRB REPORT

"I started reading it when I was a student...and I am still reading it, as a student"

—CEO, Design House



electronics FOR YOU
YOURS SINCE 1969

To Subscribe:
https://subscribe.efy.in

OR

Scan This Code

For any query, call: +91-98111-55335 or email: support@efy.in