Prometheus Monitoring

By Mohsen Mohammad Amini

Fanavaran Anisa

Iran Linux House

Linux & Open Source Training Center

cat /etc/passwd

let's all introduce ourselves to each other

(Name, Age, Education, Job, Reason/Goal)

whoami

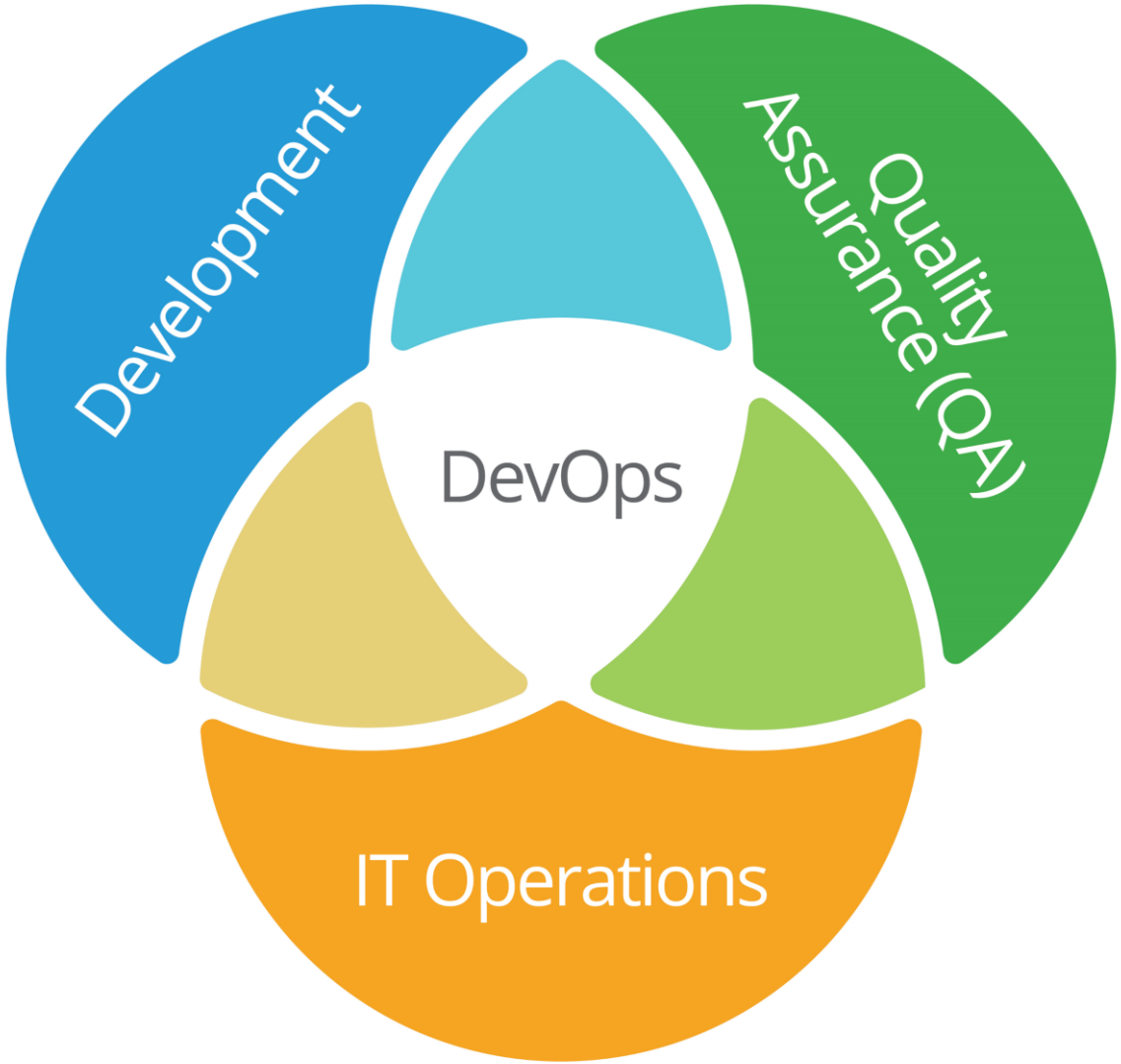
Mohsen Mohammad Amini

Email: m.mo.amini@gmail.com

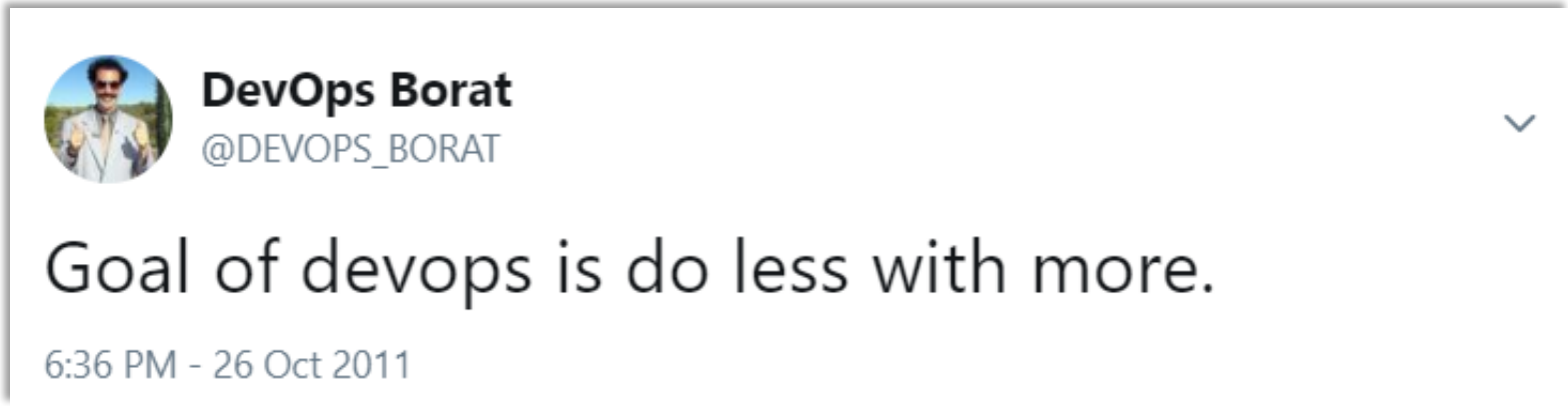
Linked In: mohsen-Mohammadamini

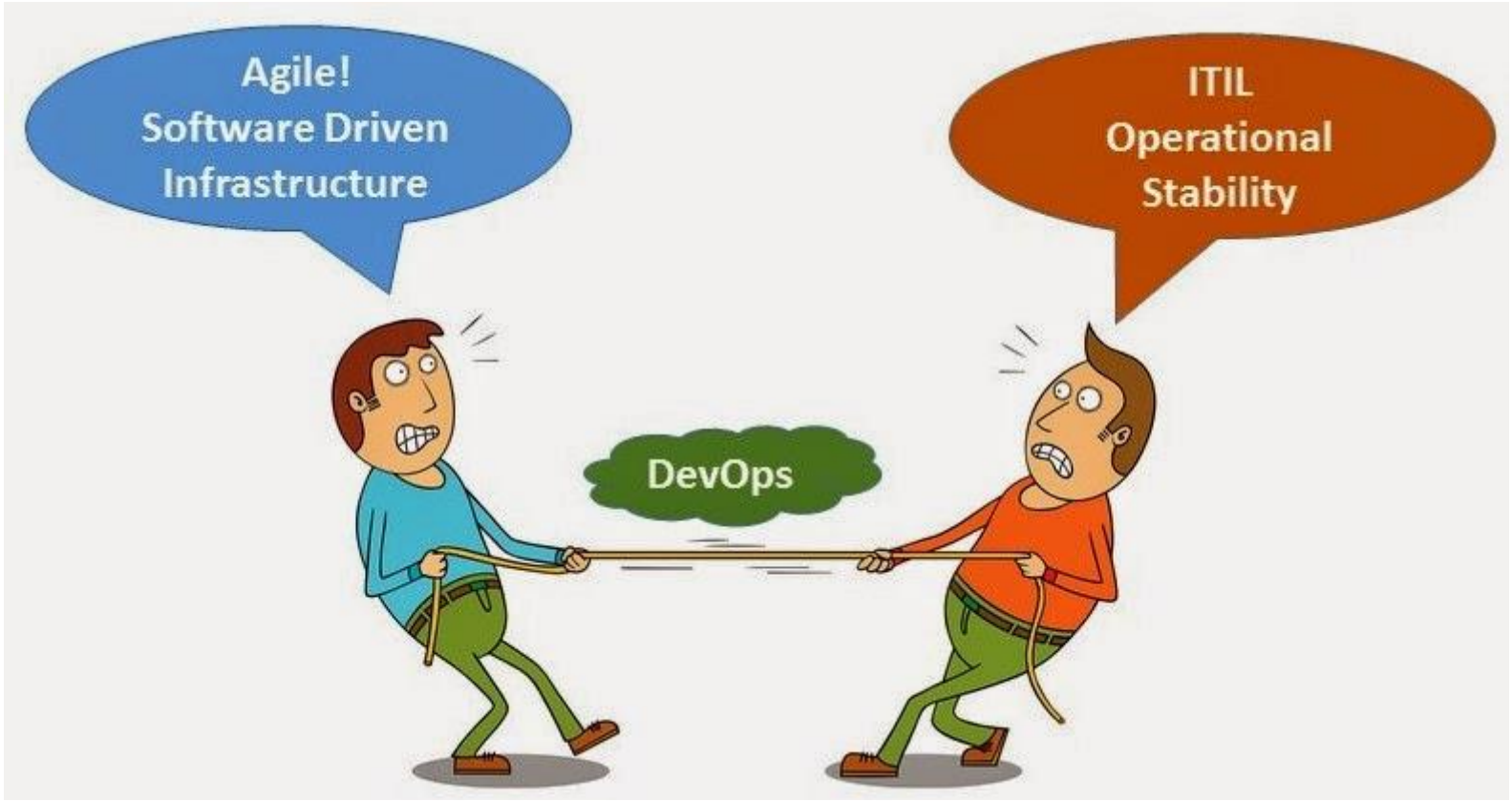
WHAT IS DEVOPS?

What is DevOps?



DevOps

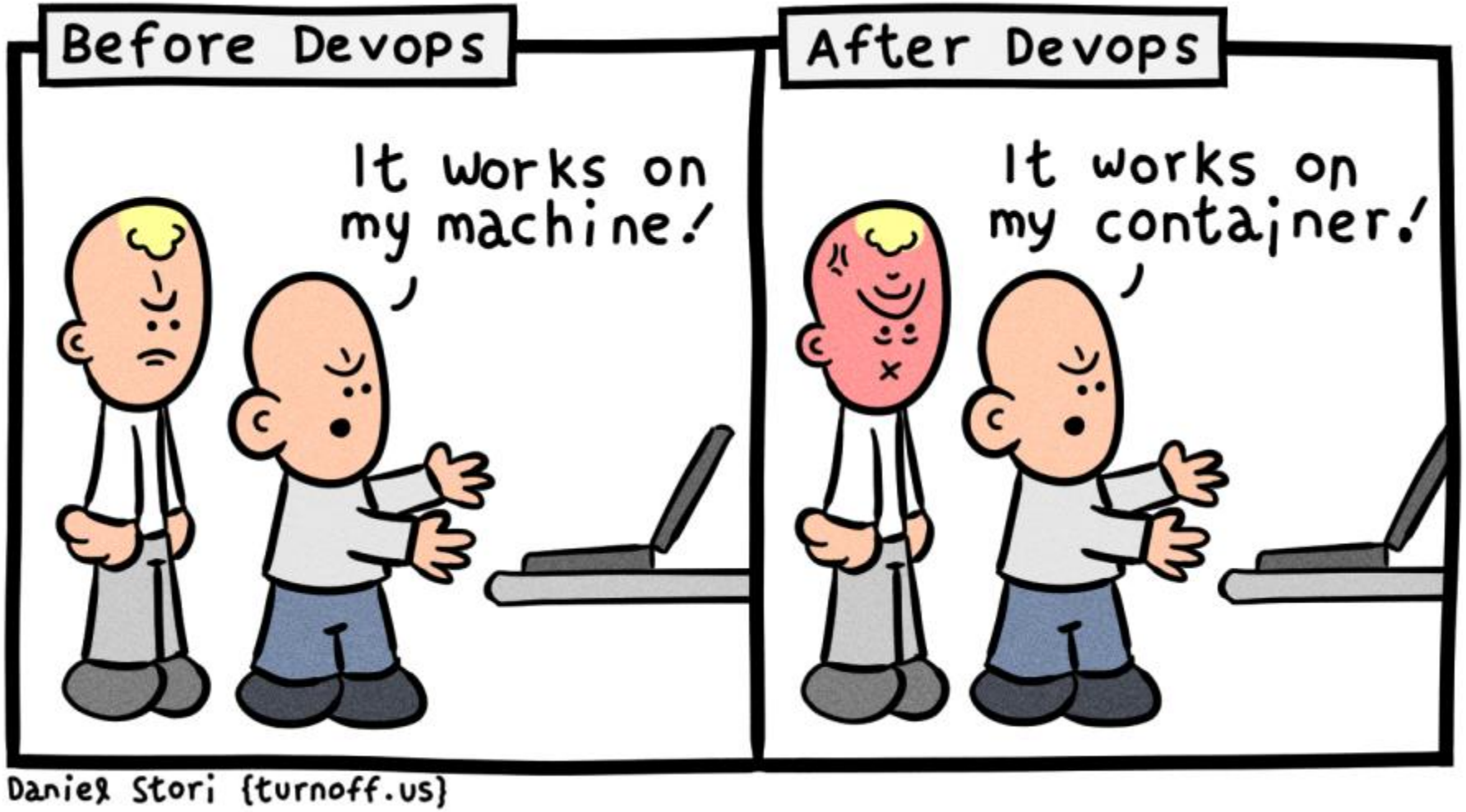


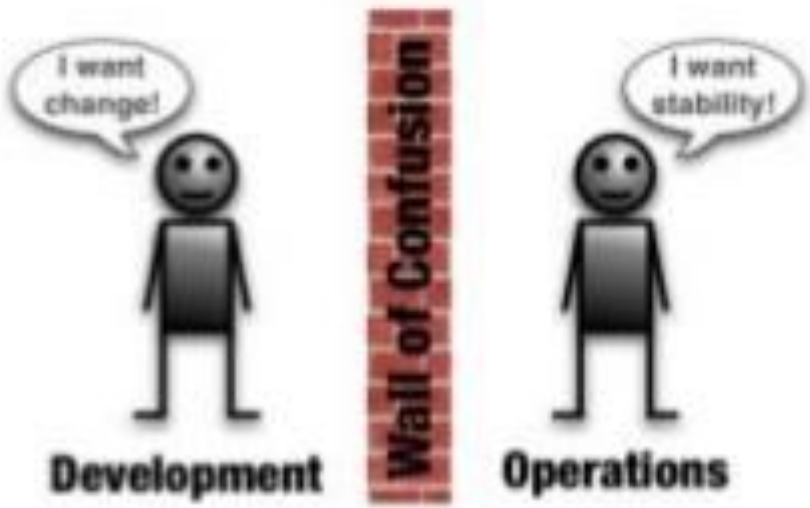



Operators
Developers


WORKED FINE IN DEV
OPS PROBLEM NOW
memegenerator.net


DevOps





old days problem

nowadays...

DevOps Definition


DevOps isn't about
automation,
just as astronomy isn't about
telescopes

What DevOps Is Not

- DevOps is not simply combining Development & Operations teams
- DevOps is not a separate team
- DevOps is not a tool
- DevOps is not a one-size-fits-all strategy
- DevOps is not automation

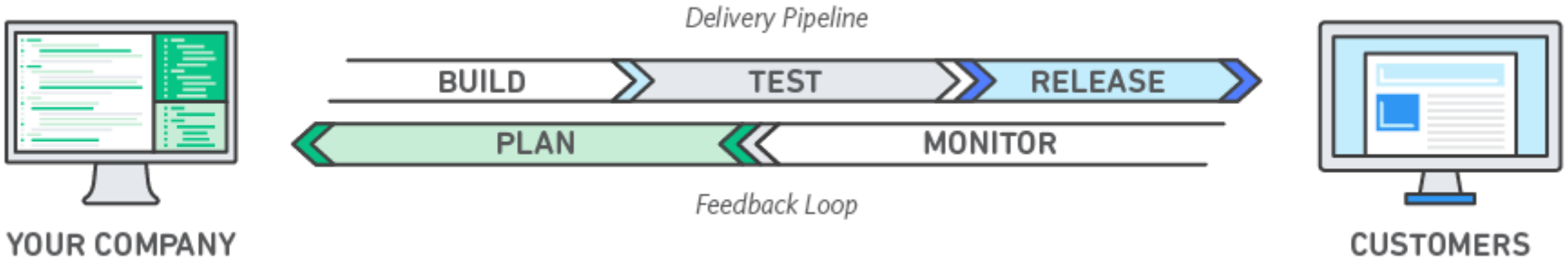
- https://devops.com/what-devops-is-not/

DevOps Model Defined

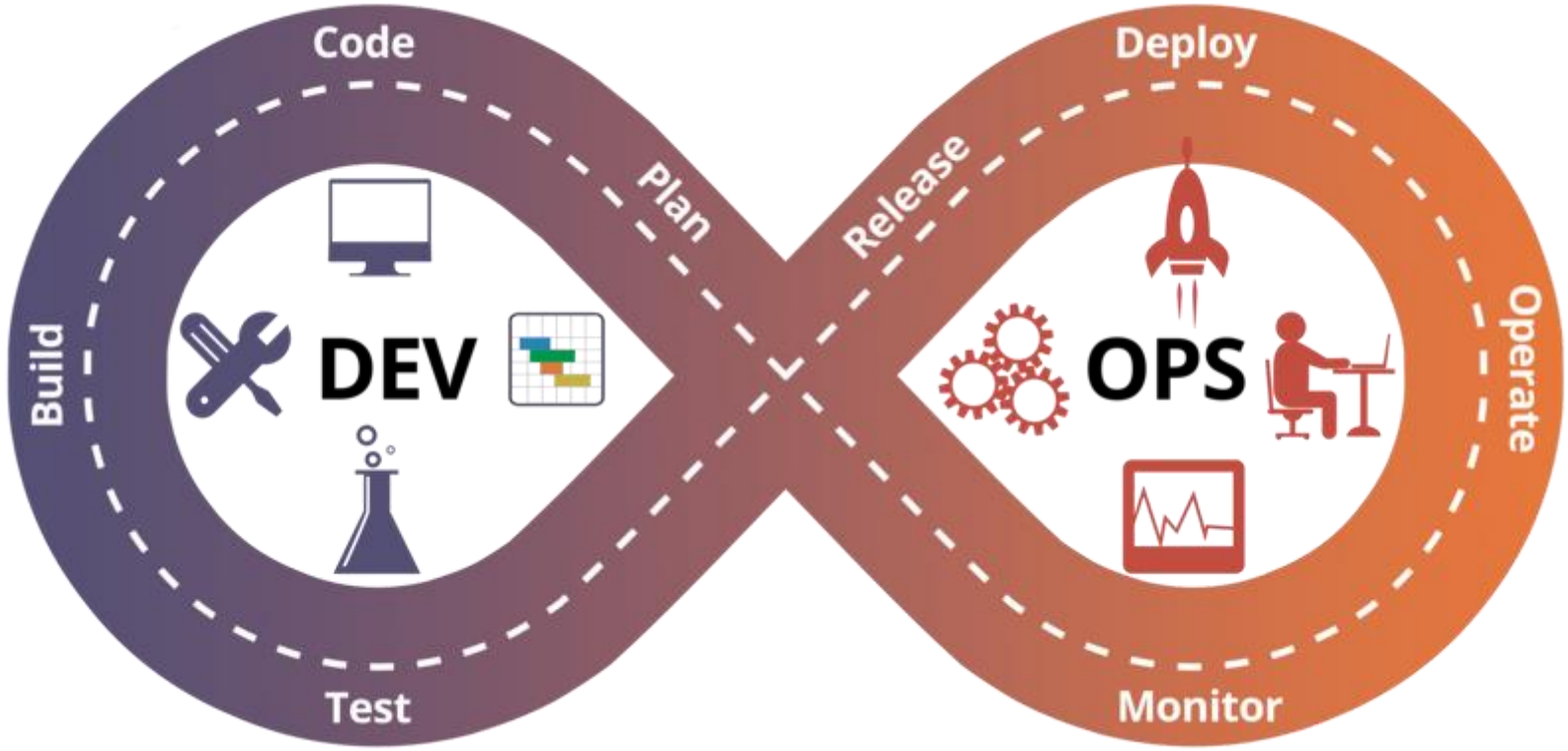
- DevOps is the combination of cultural philosophies, practices, and tools that increases an organization's ability to deliver applications and services at high velocity: evolving and improving products at a faster pace than organizations using traditional software development and infrastructure management processes.
- This speed enables organizations to better serve their customers and compete more effectively in the market.

DevOps Process

DevOps Process



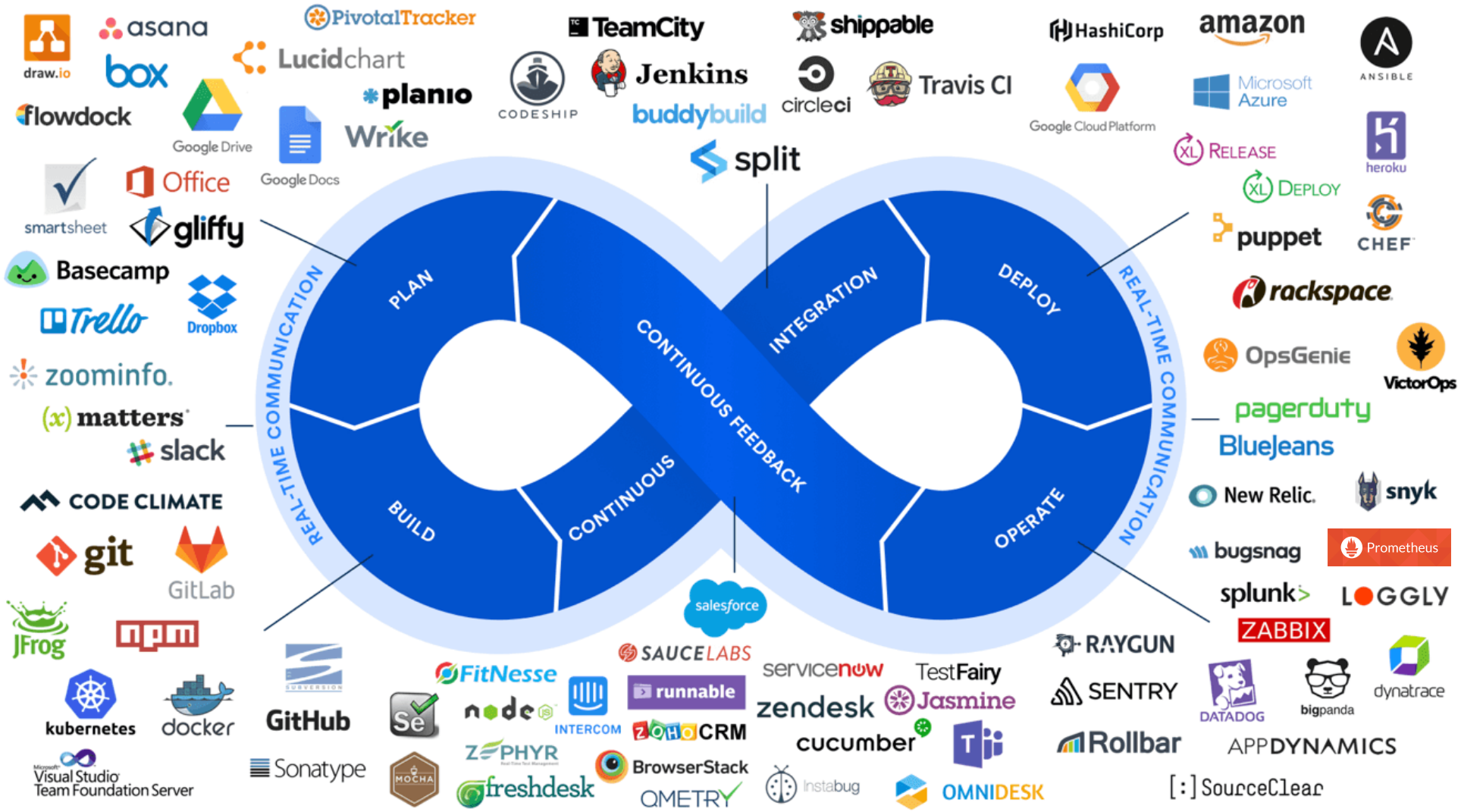
DevOps Life Cycle



What Tools Are Used in DevOps

- Source Code Repository
- Build Server
- Configuration Management
- Virtual Infrastructure
- Test Automation
- Pipeline Orchestration
- Monitoring Systems
- Team Collaboration

DevOps Toolchain



DevOps Goals

- **Improve collaboration between all stakeholders from planning through delivery and automation of the delivery process in order to:**
  - ✓ Improve deployment frequency
  - ✓ Achieve faster time to market
  - ✓ Lower failure rate of new releases
  - ✓ Shorten lead time between fixes
  - ✓ Improve mean time to recovery

DevOps Values

- It’s important to keep CALMS
- Culture: People > Process> Tools
- Automation: Infrastructure as Code
- Lean: Focus on value and customer
- Measurement: Measure everything
- Sharing: Collaboration / Feedback

DevOps Practices

Continuous Integration

Continuous Delivery

Microservices

Infrastructure as Code

Monitoring and Logging

Communication and Collaboration

Continuous Integration

- Continuous integration is a DevOps software development practice where developers regularly merge their code changes into a central repository, after which automated builds and tests are run.

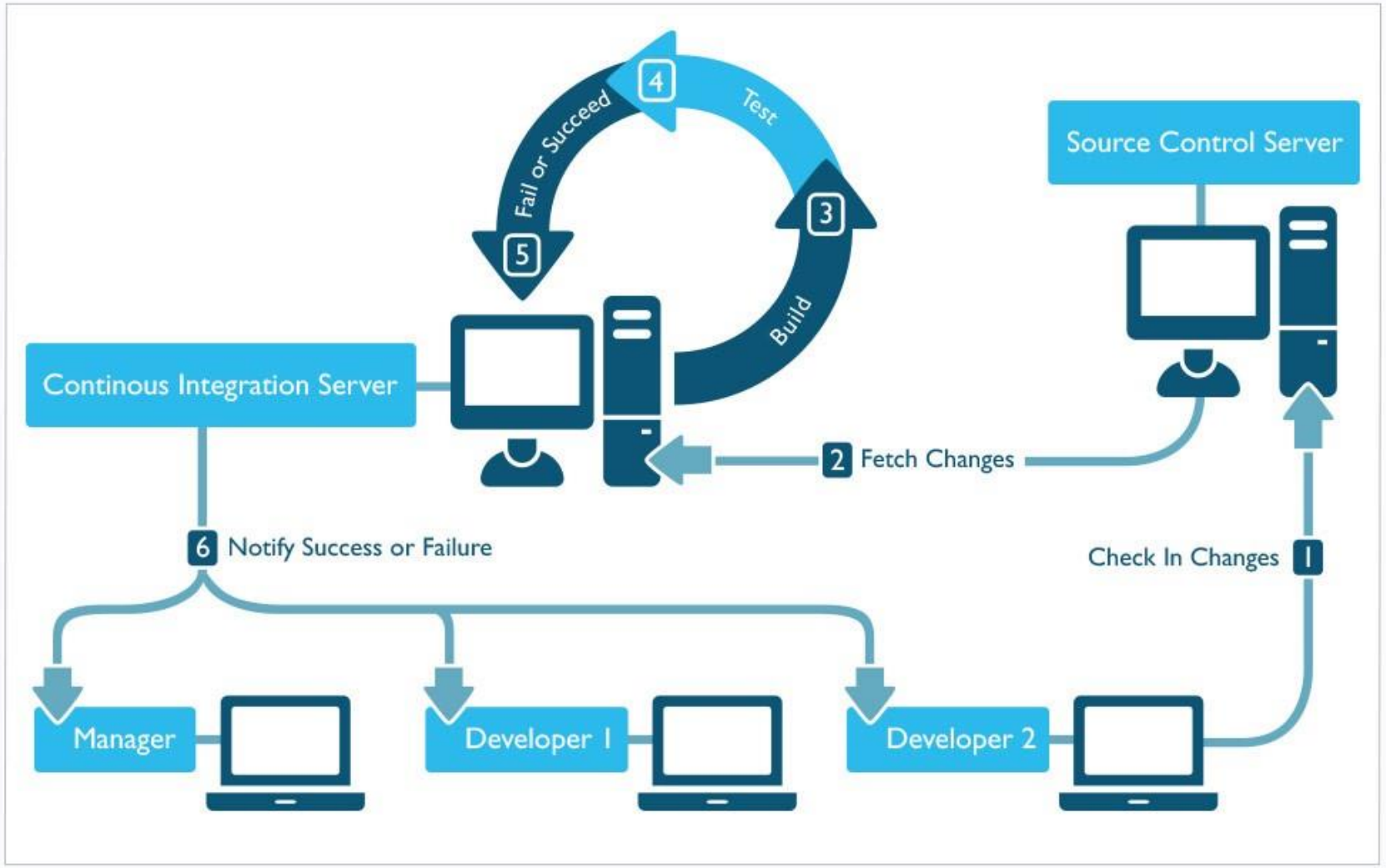
Continuous Integration

- Continuous integration most often refers to the build or integration stage of the software release process and entails both an automation component (e.g. a CI or build service) and a cultural component (e.g. learning to integrate frequently).
- The key goals of continuous integration are to find and address bugs quicker, improve software quality, and reduce the time it takes to validate and release new software updates.

Continuous Integration

- With continuous integration, developers frequently commit to a shared repository using a version control system such as Git.
- Prior to each commit, developers may choose to run local unit tests on their code as an extra verification layer before integrating.
- A continuous integration service automatically builds and runs unit tests on the new code changes to immediately surface any errors.

Continuous Integration



Continuous Delivery

- Continuous delivery is a software development practice where code changes are automatically prepared for a release to production.
- A pillar of modern application development, continuous delivery expands upon continuous integration by deploying all code changes to a testing environment and/or a production environment after the build stage.
- When properly implemented, developers will always have a deployment-ready build artifact that has passed through a standardized test process.

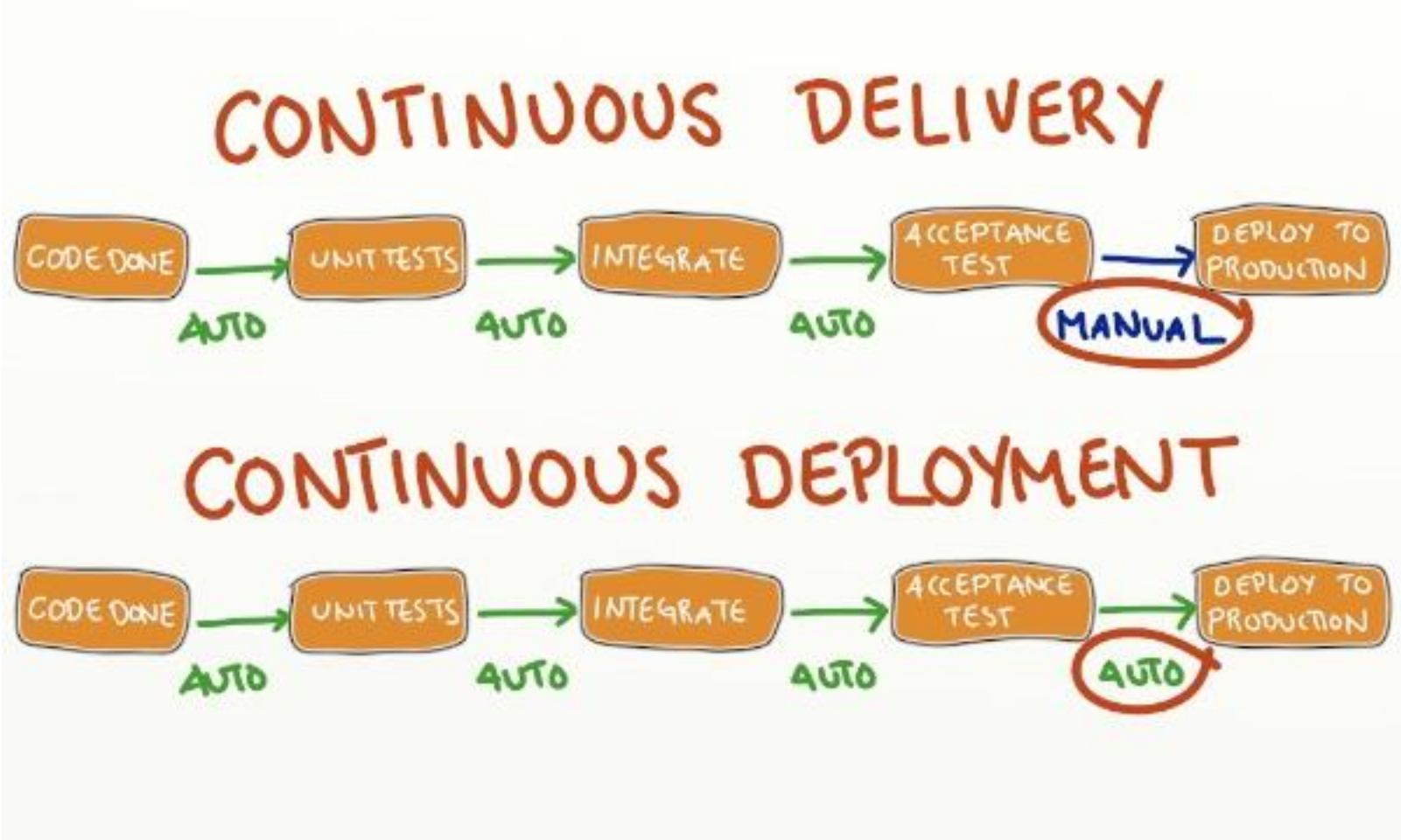
Continuous Delivery

- Continuous delivery lets developers automate testing beyond just unit tests so they can verify application updates across multiple dimensions before deploying to customers.
- These tests may include UI testing, load testing, integration testing, API reliability testing, etc.
- This helps developers more thoroughly validate updates and pre-emptively discover issues. With the cloud, it is easy and cost-effective to automate the creation and replication of multiple environments for testing, which was previously difficult to do on-premises.

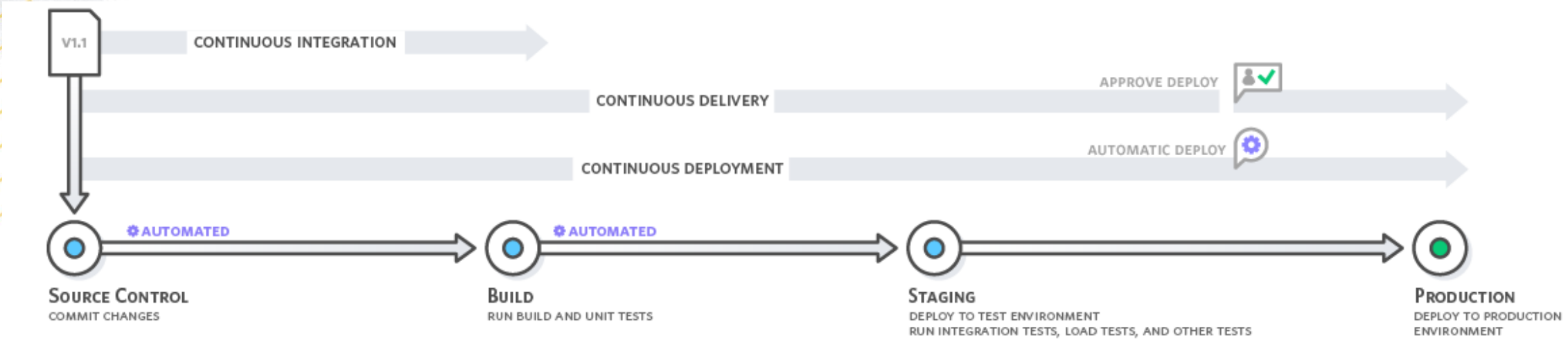
Continuous Delivery vs. Continuous Deployment

- With continuous delivery, every code change is built, tested, and then pushed to a non-production testing or staging environment. There can be multiple, parallel test stages before a production deployment.
- The difference between continuous delivery and continuous deployment is the presence of a manual approval to update to production.
- With continuous deployment, production happens automatically without explicit approval.

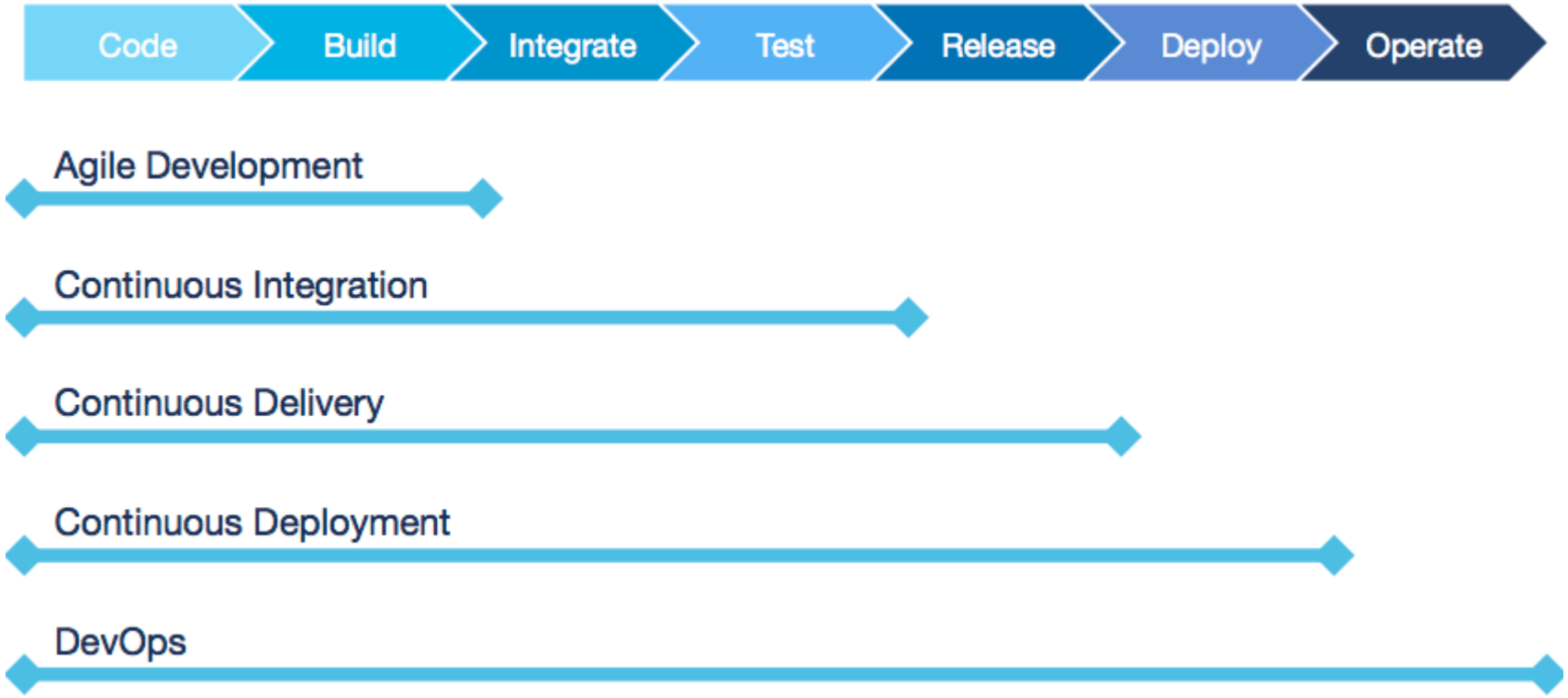
Continuous Delivery vs. Continuous Deployment



CI/CD



Agile/CI/CD/DevOps



Prometheus

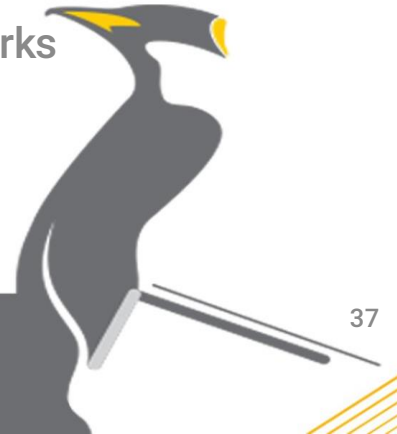
Chapter Overview

- **We will be covering the following topics in this chapter:**
  - ✓ Defining of monitoring
  - ✓ Whitebox versus Blackbox monitoring
  - ✓ Understanding metrics collection

What Is Monitoring?

There is no consensus on what exactly Monitoring means.

Generally, Monitoring may aid software developers and administrators in the operation of production computer systems, such as the applications, tools, databases, and networks backing popular websites.



Organizational Contexts

- Looking into an organizational context, roles such as system administrators, quality assurance engineers, Site Reliability Engineers (SREs), or product owners have different expectations from monitoring.

System Administrators

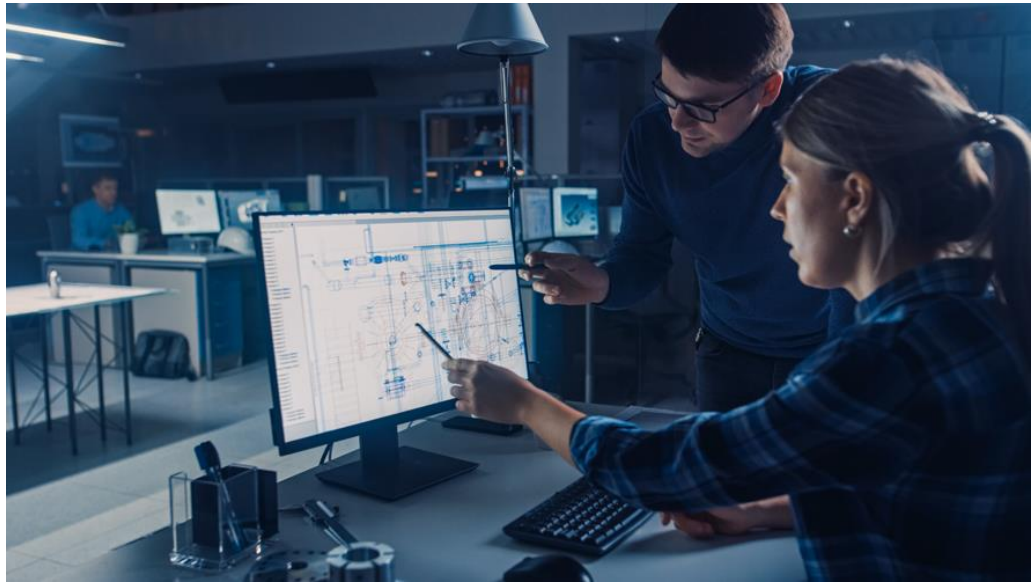
- **High-resolution**: for a system administrator, the main objective of monitoring is to obtain visibility across the infrastructure and manage data, so that problems are quickly discovered and the root causes are identified as soon as possible.
- **Low-latency**: if a problem is occurring, you don't have the privilege to wait several hours for your next data point, and so data has to be provided in near real time.
- **High-diversity data**: since there is no easy way to identify or predict which systems are prone to be affected, we need to collect as much data as possible from all systems

Quality Assurance Engineers

- **High-resolution**: high resolution monitoring data collected, which enables a deeper drill down into effects,
- **High latency**: the latency is not as critical as it is for system administrators.
  - ✓ In this case, historical data is much more critical for comparing software releases than the freshness of the data.
- **High-diversity**: since we can't wholly predict the ramifications of a new release, the available data needs to be spread across as much of the infrastructure as possible, touching every system the software release might use and invoke it or generally interact with it (directly or indirectly), so that we have as much data as possible.

SRE (Site Reliability Engineers)

- Interested in **low-resolution**, **high-latency**, and **high-diversity** data.
- In this scenario, historical data carries much more importance for SREs than the resolution that this data is presented in.
- As such, it is also important for SREs to have a broad visualization of all the different parts of the infrastructure that are affected by those requirements to predict, for example, the amount of storage for logs, network bandwidth increase, and so on, making the high diversity of monitoring data mandatory



Monitoring Components

- **Alerting**
  - ✓ Knowing when things are going wrong is usually the most important thing that you want monitoring for.
  - ✓ You want the monitoring system to call in a human to take a look.
  - ✓ This is the continuous threshold validation of metrics or logs, and fires an action or notification in the case of a transgression of the said threshold
- **Debugging**
  - ✓ Now that you have called in a human, they need to investigate to determine the root cause and ultimately resolve whatever the issue is.
- **Trending**
  - ✓ Alerting and debugging usually happen on time scales on the order of minutes to hours.
  - ✓ While less urgent, the ability to see how your systems are being used and changing over time is also useful.
  - ✓ Trending can feed into design decisions and processes such as capacity planning.
- **Plumbing**
  - ✓ At the end of the day all monitoring systems are data processing pipelines.
  - ✓ Sometimes it is more convenient to appropriate part of your monitoring system for another purpose, rather than building a bespoke solution.

Categories of Monitoring

- **Metrics**
  - ✓ This exposes a certain system resource, application action, or business characteristic as a specific point in time value.
  - ✓ This information is obtained in an aggregated form.
- **Logging**
  - ✓ Containing much more data than a metric, this manifests itself as an event from a system or application, containing all the information that's produced by such an event.
  - ✓ This information is not aggregated and has the full context.
- **Tracing:**
  - ✓ This is a special case of logging where a request is given a unique identifier so that it can be tracked during its entire life cycle across every system.
  - ✓ Due to the increase of the dataset with the number of requests, it is a good idea to use samples instead of tracking all requests.
- **Visualization:**
  - ✓ This is a graphical representation of metrics, logs, or traces.

Whitebox vs Blackbox Monitoring

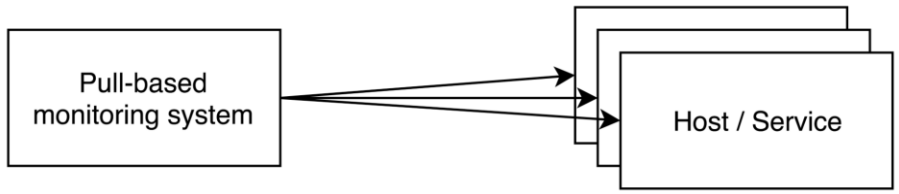
- In **Blackbox** monitoring, the application or host is observed from the outside and, consequently, this approach can be fairly limited.
- Checks are made to assess whether the system under observation responds to probes in a known way:
  - ✓ Does the host respond to **ICMP** echo requests?
  - ✓ Is a given **TCP** port open?
  - ✓ Does the application respond with the correct data and **status code** when it receives a specific HTTP request?
  - ✓ Is the process for a specific application **running** in its host?

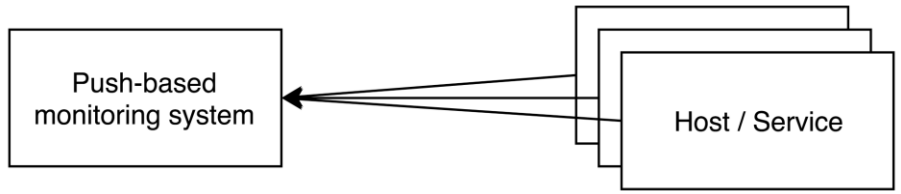
Whitebox vs Blackbox Monitoring

- In **Whitebox** monitoring, the system under observation surfaces data about its internal state and the performance of critical sections.
- This type of introspection can be very powerful as it exposes the operating telemetry, and consequently the health, of the different internal components, which otherwise would be difficult or even impossible to ascertain.
- This telemetry data is usually handled in the following ways:
  - ✓ Exported through logging
  - ✓ Emitted as structured events
  - ✓ Maintained in memory as aggregates

Understanding metrics collection

- The process by which metrics are by monitoring systems can generally be divided into two approaches: **push** and **pull**.





Push vs Pull

- **In push-based systems, the monitored hosts and services make themselves known by reporting to the monitoring system.**
  - ✓ The advantage here is that no prior knowledge of new systems is required for them to be picked up.
  - ✓ However, this means that the monitoring service's location needs to be propagated to all targets, usually with some form of configuration management.
  - ✓ Staleness is a big drawback of this approach: if a system hasn't reported in for some time, does that mean it's having problems or was it purposely decommissioned?
  - ✓ Furthermore, when you manage a distributed fleet of hosts and services that push data to a central point, the risk of a **thundering herd** (overload due to many incoming connections at the same time) or a misconfiguration causing an unforeseen flood of data becomes much more complex and time-consuming to mitigate.

Push vs Pull

- **In pull-based monitoring, the system needs a definitive list of hosts and services to monitor so that their metrics are ingested.**
  - ✓ Having a central source of truth provides some level of assurance that everything is where it's supposed to be, with the **drawback** of having to maintain said source of truth and keeping it updated with any changes.
  - ✓ With the rapid rate of change in today's infrastructures, some form of automated discovery is needed to keep up with the full picture.
  - ✓ Having a centralized point of configuration enables a much faster response in the case of issues or misconfigurations.

Choosing A Monitoring Tool

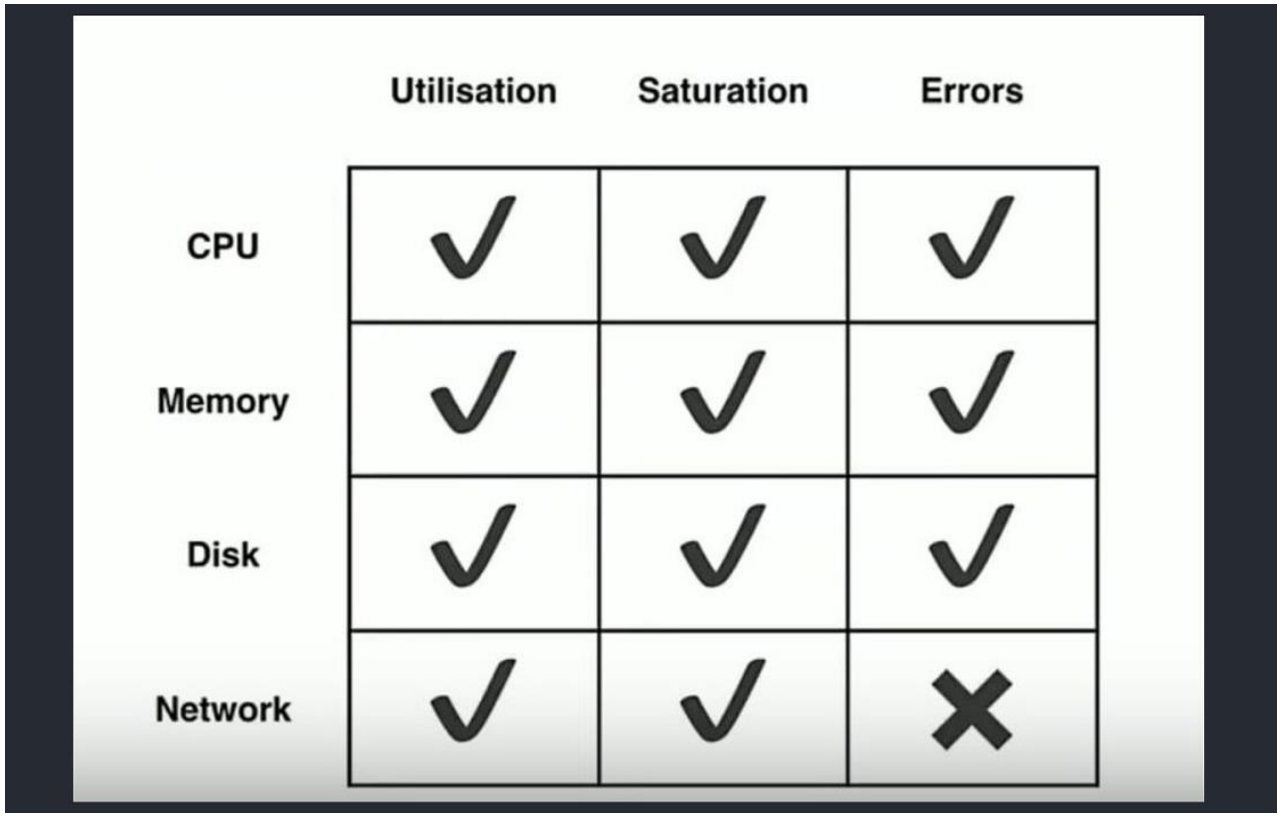
- There are other more important factors when choosing a monitoring tool, such as **flexibility**, **ease of automation**, **maintainability**, or **broad support for the technologies being used**.
- Even though Prometheus is a **pull-based** monitoring system, it also provides a way of ingesting pushed metrics by using a gateway that converts from push to pull.

What to measure

- Google's four golden signals

What to measure

- Brendan Gregg's USE method
  - ✓Brendan's method is more machine-focused



What to measure

- **Tom Wilkie's RED method**
  - ✓ The RED method is more focused on a service-level approach and not so much on the underlying system itself
    - **Rate**: Translated as requests per second
    - **Errors**: The amount of failing requests per second
    - **Duration**: The time taken by those requests

WHAT IS PROMETHEUS?

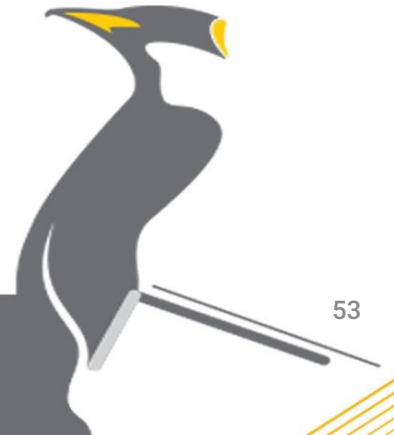
Prometheus is an open source, metrics-based monitoring system.

Prometheus does one thing and it does it well.

Since its beginnings with no more than a handful of developers working in SoundCloud in 2012, a community and ecosystem has grown round Prometheus.

Prometheus is primarily written in Go and licensed under the Apache 2.0 license.

In 2016 the Prometheus project became the second member of the Cloud Native Computing Foundation (CNCF).



What Is Prometheus?

- It has a simple yet powerful data model and a query language that lets you analyze how your applications and infrastructure are performing.
- **Software like Kubernetes and Docker are already instrumented with Prometheus client libraries.**
- For third-party software that exposes metrics in a non-Prometheus format, there are hundreds of integrations available.
  - ✓ These are called **exporters**, and include HAProxy, MySQL, PostgreSQL, Redis, JMX, SNMP, Consul, and Kafka.
- For instrumenting your own code, there are client libraries in all the popular languages and runtimes, including **Go**, **Java/JVM**, **C#/.Net**, **Python**, **Ruby**, **Node.js**, **Haskell**, **Erlang**, and **Rust**.

What Is Prometheus?

- A simple text format makes it easy to expose metrics to Prometheus.
  - ✓ Other monitoring systems, both open source and commercial, have added support for this format.
- The data model identifies each time series not just with a name, but also with an unordered set of key-value pairs called labels.
- The PromQL query language allows aggregation across any of these labels, so you can analyze not just per process but also per datacenter and per service or by any other labels that you have defined.
- These can be graphed in dashboard systems such as Grafana.
- Alerts can be defined using the exact same PromQL query language that you use for graphing.
  - ✓ If you can graph it, you can alert on it.
- Labels make maintaining alerts easier, as you can create a single alert covering all possible label values.
- Relatedly, service discovery can automatically determine what applications and machines should be scraped from sources such as Kubernetes, Consul, Amazon Elastic Compute Cloud (EC2), Azure, Google Compute Engine (GCE), and OpenStack.

What Is Prometheus?

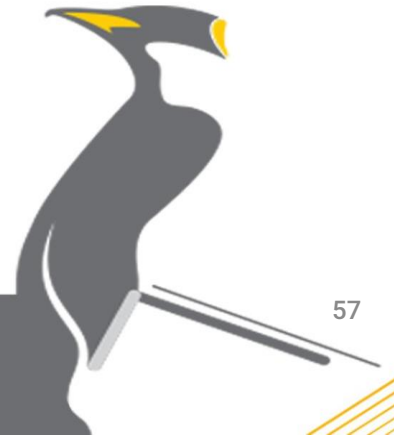
- **For all these features and benefits, Prometheus is performant and simple to run.**
  - ✓ A single Prometheus server can ingest millions of samples per second.
  - ✓ It is a single statically linked binary with a configuration file.
  - ✓ All components of Prometheus can be run in containers, and they avoid doing anything fancy that would get in the way of configuration management tools.
  - ✓ It is designed to be integrated into the infrastructure you already have and built on top of, not to be a management platform itself.

PROMETHEUS ARCHITECTURE

Prometheus discovers targets to scrape from service discovery.

These can be your own instrumented applications or third-party applications you can scrape via an exporter.

The scraped data is stored, and you can use it in dashboards using PromQL or send alerts to the Alert manager, which will convert them into pages, emails, and other notifications.



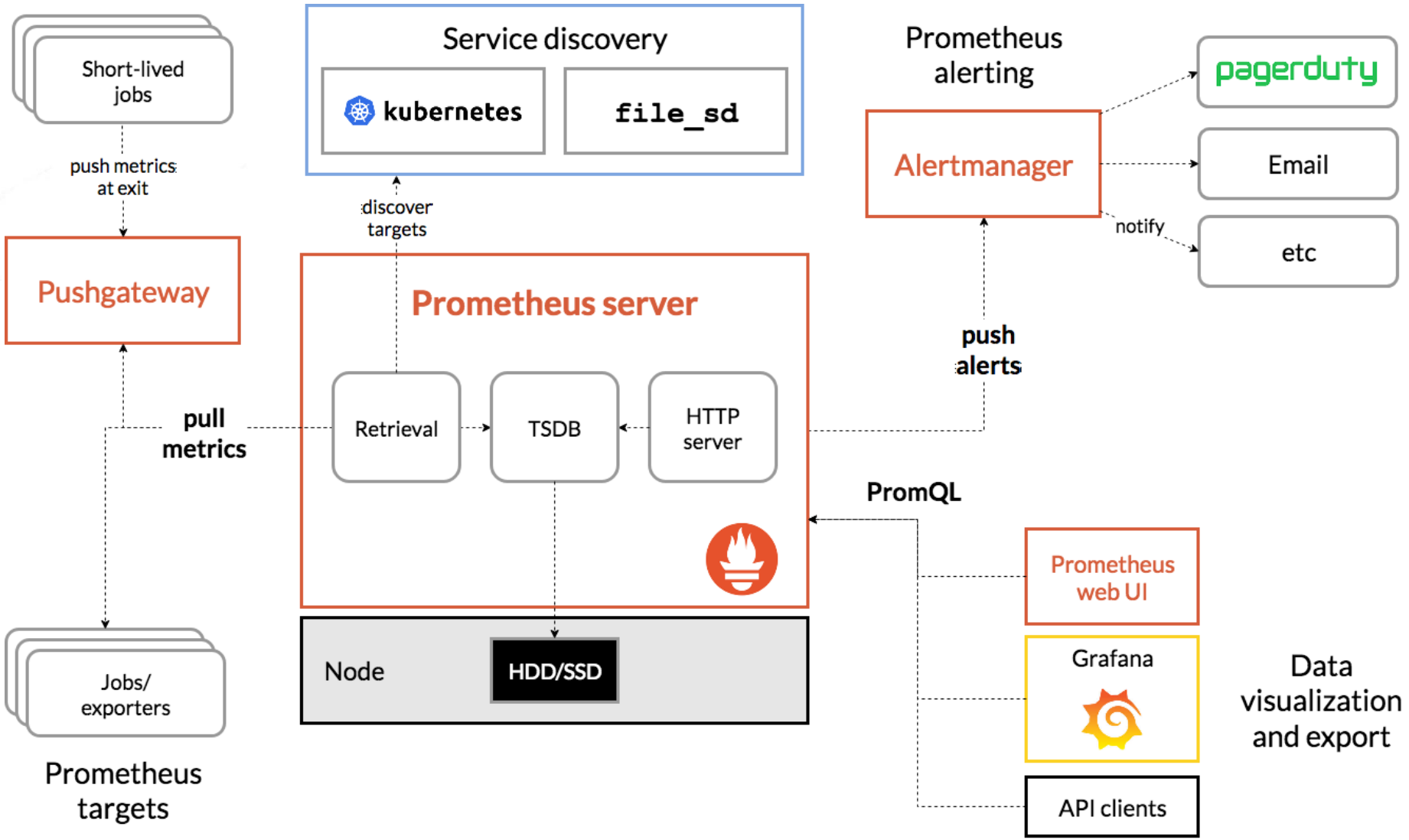
Prometheus Ecosystem

- Striving for simplicity and having a clear understanding of all the moving parts of a Prometheus stack is invaluable to keep things manageable and reliable.
- In brief, the following topics will be covered in this chapter:
  - ✓ Metrics collection with Prometheus
  - ✓ Exposing internal state with exporters
  - ✓ Alert routing and management with Alertmanager
  - ✓ Visualizing your data

Metrics collection with Prometheus

- Prometheus is a time series-based, open source monitoring system.
- It collects data by sending HTTP requests to hosts and services on metrics endpoints, which it then makes available for analysis and alerting using a powerful query language.

Prometheus Architecture



Prometheus Architecture

- The **Prometheus server** collects time series data, stores it, makes it available for querying, and sends alerts based on it.
- The **Alertmanager** receives alert triggers from Prometheus and handles routing and dispatching of alerts.
- The **Pushgateway** handles the exposition of metrics that have been pushed from short-lived jobs such as cron or batch jobs.
- **Applications** that support the Prometheus exposition format make internal state available through an HTTP endpoint.
- **Community-driven** exporters expose metrics from applications that do not support Prometheus natively.
- First-party and third-party dashboarding solutions provide a **visualization** of collected data.

Client Libraries

- Metrics do not typically magically spring forth from applications; someone has to add the instrumentation that produces them.
  - ✓ This is where client libraries come in.
- With usually only two or three lines of code, you can both define a metric and add your desired instrumentation inline in code you control.
  - ✓ This is referred to as direct instrumentation.
- Client libraries are available for all the major languages and runtimes.
- The Prometheus project provides official client libraries in Go, Python, Java/JVM, and Ruby.
- There are also a variety of third-party client libraries, such as for C#/.Net, Node.js, Haskell, Erlang, and Rust.
- Client libraries take care of all the nitty-gritty details such as thread-safety, bookkeeping, and producing the Prometheus text exposition format in response to HTTP requests.

Client Libraries

- As metrics-based monitoring does not track individual events, client library memory usage does not increase the more events you have.
  - ✓ Rather, memory is related to the number of metrics you have.
- If one of the library dependencies of your application has Prometheus instrumentation, it will automatically be picked up.
- Thus by instrumenting a key library such as your RPC client, you can get instrumentation for it in all of your applications.
- Some metrics are typically provided out of the box by client libraries such as CPU usage and garbage collection statistics, depending on the library and runtime environment.
- Client libraries are not restricted to outputting metrics in the Prometheus text format.
- Prometheus is an open ecosystem, and the same APIs used to feed the generation text format can be used to produce metrics in other formats or to feed into other instrumentation systems.
- Similarly, it is possible to take metrics from other instrumentation systems and plumb it into a Prometheus client library, if you haven't quite converted everything to Prometheus instrumentation yet.

Exporters

- Not all code you run is code that you can control or even have access to, and thus adding direct instrumentation isn't really an option.
- An exporter is a piece of software that you deploy right beside the application you want to obtain metrics from.
- It takes in HTTP requests from Prometheus, gathers the required data from the application, transforms them into the correct format, and finally returns them in a response to Prometheus.
- Each exporter usually targets a specific service or application and as such, their deployment reflects this one-to-one synergy.
- If the exporter is missing a metric you are interested in, you can always send a pull request to improve it, making it better for the next person to use it.

Exporter fundamentals

- When the exporter starts, it binds to a configured port and exposes the internal state of whatever is being collected in an HTTP endpoint of your choosing (the default being `/metrics`).
- The instrumentation data is collected when an HTTP GET request is made to the configured endpoint.
- The HTTP GET request that's made by the Prometheus server to the observed system for metric collection is called a **scrape**.

Service Discovery

- Once you have all your applications instrumented and your exporters running, Prometheus needs to know where they are.
- This is so Prometheus will know what is meant to monitor, and be able to notice if something it is meant to be monitoring is not responding.
- With dynamic environments you cannot simply provide a list of applications and exporters once, as it will get out of date.
  - ✓ This is where service discovery comes in.
- Prometheus has integrations with many common service discovery mechanisms, such as Kubernetes, EC2, and Consul.
- There is also a generic integration for those whose setup is a little off the beaten path.
- As every organization does it slightly differently, Prometheus allows you to configure how metadata from service discovery is mapped to monitoring targets and their labels using relabelling.

Scraping

- Service discovery and relabelling give us a list of targets to be monitored.
  - ✓ Now Prometheus needs to fetch the metrics.
- Prometheus does this by sending a HTTP request called a scrape.
  - ✓ The response to the scrape is parsed and ingested into storage.
- Several useful metrics are also added in, such as if the scrape succeeded and how long it took.
- Scrapes happen regularly; usually you would configure it to happen every 10 to 60 seconds for each target.

Storage

- Prometheus stores data locally in a custom database.
- Distributed systems are challenging to make reliable, so Prometheus does not attempt to do any form of clustering.
  - ✓ In addition to reliability, this makes Prometheus easier to run.
- Over the years, storage has gone through a number of redesigns, with the storage system in Prometheus 2.0 being the third iteration.
- The storage system can handle ingesting millions of samples per second, making it possible to monitor thousands of machines with a single Prometheus server.
- The compression algorithm used can achieve 1.3 bytes per sample on real-world data.
- An SSD is recommended, but not strictly required.

Dashboards

- Prometheus has a number of HTTP APIs that allow you to both request raw data and evaluate PromQL queries.
- These can be used to produce graphs and dashboards.
- Out of the box, Prometheus provides the expression browser.
  - ✓ It uses these APIs and is suitable for ad hoc querying and data exploration, but it is not a general dashboard system.
- It is recommended that you use Grafana for dashboards.
  - ✓ It has a wide variety of features, including official support for Prometheus as a data source.

Recording Rules and Alerts

- Although PromQL and the storage engine are powerful and efficient, aggregating metrics from thousands of machines on the fly every time you render a graph can get a little laggy.
- Recording rules allow PromQL expressions to be evaluated on a regular basis and their results ingested into the storage engine.
- Alerting rules are another form of recording rules.
- They also evaluate PromQL expressions regularly, and any results from those expressions become alerts.
- Alert are sent to the Alertmanager.

Alert Management

- The Alertmanager receives alerts from Prometheus servers and turns them into notifications.
  - ✓ Notifications can include email, chat applications such as Slack, and serv ices such as PagerDuty.
- The Alertmanager does more than blindly turn alerts into notifications on a one-to-one basis.
- Related alerts can be aggregated into one notification, throttled to reduce pager storms, and different routing and notification outputs can be configured for each of your different teams.
- Alerts can also be silenced, perhaps to snooze an issue you are already aware of in advance when you know maintenance is scheduled.
- The Alertmanager's role stops at sending notifications; to manage human responses to incidents you should use services such as PagerDuty and ticketing systems.

Long-Term Storage

- Since Prometheus stores data only on the local machine, you are limited by how much disk space you can fit on that machine.
- While you usually care only about the most recent day or so worth of data, for long-term capacity planning a longer retention period is desirable.
- Prometheus does not offer a clustered storage solution to store data across multiple machines, but there are remote read and write APIs that allow other systems to hook in and take on this role.
- These allow PromQL queries to be transparently run against both local and remote data.

WHAT PROMETHEUS IS NOT

Now that you have an idea of where Prometheus fits in the broader monitoring landscape and what its major components are, let's look at some use cases for which Prometheus is not a particularly good choice.

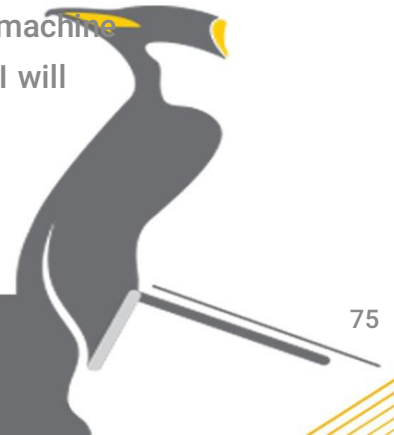
What Prometheus Is Not

- As a metrics-based system, Prometheus is not suitable for storing event logs or individual events.
- Nor is it the best choice for high cardinality data, such as email addresses or usernames.
- Prometheus is designed for operational monitoring, where small inaccuracies and race conditions due to factors like kernel scheduling and failed scrapes are a fact of life.
- Prometheus makes tradeoffs and prefers giving you data that is 99.9% correct over your monitoring breaking while waiting for perfect data.
- Thus in applications involving money or billing, Prometheus should be used with caution.

Getting Started with Prometheus

In this chapter you will set up and run Prometheus, the Node exporter, and the Alertmanager. This simple example will monitor a single machine and give you a small taste of what a full Prometheus deployment looks like. Later chapters will look at each aspect of this setup in detail.

This chapter requires a machine running any reasonable, modern version of Linux. Either bare metal or a virtual machine will do. You will use the command line and access services on the machine using a web browser. For simplicity I will assume that everything is running on localhost; if this is not the case, adjust the URLs as appropriate.



Prometheus Installation Ingredients

- VirtualBox >= 6.1.10
- CentOS 7 OVA file
- Putty and MTputty for Windows students
- SSH client for Linux/Unix students

Running Prometheus

```
# hostnamectl set-hostname Prometheus-server
# timedatectl set-timezone Asia/Tehran
# yum install wget vim
# wget \
https://github.com/prometheus/prometheus/releases/download/
v2.25.0-rc.0/prometheus-2.25.0-rc.0.linux-amd64.tar.gz
```

Running Prometheus

```
# useradd --no-create-home --shell /bin/false prometheus
# mkdir /etc/prometheus
# mkdir /var/lib/prometheus
# chown prometheus:prometheus /etc/prometheus
# chown prometheus:prometheus /var/lib/prometheus
```

Running Prometheus

```
# tar -xzf prometheus-*.linux-amd64.tar.gz

# cd prometheus-*.linux-amd64/

# cp prometheus /usr/local/bin/

# cp promtool /usr/local/bin/

# chown prometheus:prometheus /usr/local/bin/prometheus

# chown prometheus:prometheus /usr/local/bin/promtool
```

Running Prometheus

```
# cp -r consoles /etc/prometheus
# cp -r console_libraries /etc/prometheus
# chown -R prometheus:prometheus /etc/prometheus/consoles
# chown -R prometheus:prometheus /etc/prometheus/console_libraries
# vim /etc/prometheus/prometheus.yml
global:
  scrape_interval: 10s
scrape_configs:
  - job_name: Prometheus_master
    static_configs:
      - targets: ['localhost:9090']
```

Running Prometheus

```
# vim /etc/systemd/system/prometheus.service
[Unit]
Description=Prometheus
Wants=network-online.target
After=network-online.target

[Service]
User=prometheus
Group=prometheus
Type=simple
ExecStart=/usr/local/bin/prometheus \
--config.file /etc/prometheus/prometheus.yml \
--storage.tsdb.path /var/lib/prometheus/ \
--web.console.templates=/etc/prometheus/consoles \
--web.console.libraries=/etc/prometheus/console_libraries

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

Running Prometheus

```
# systemctl daemon-reload
# systemctl enable --now prometheus
# systemctl status prometheus

# firewall-cmd --add-port=9090/tcp --permanent
# firewall-cmd --reload
```