iTECH.
HR, Learning & Development

ribbon

Welcome to the iTECH webinar

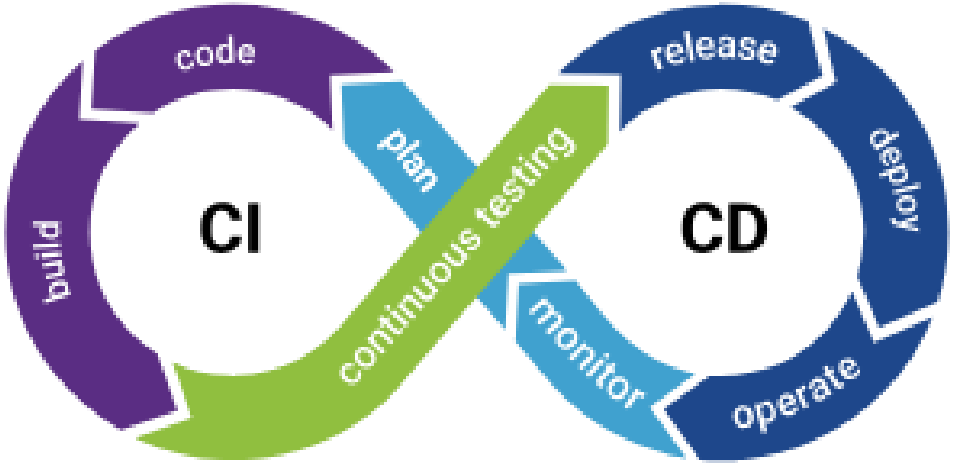
Container-based Solution, CI/CD, Automation

By

Samuel Dratwa

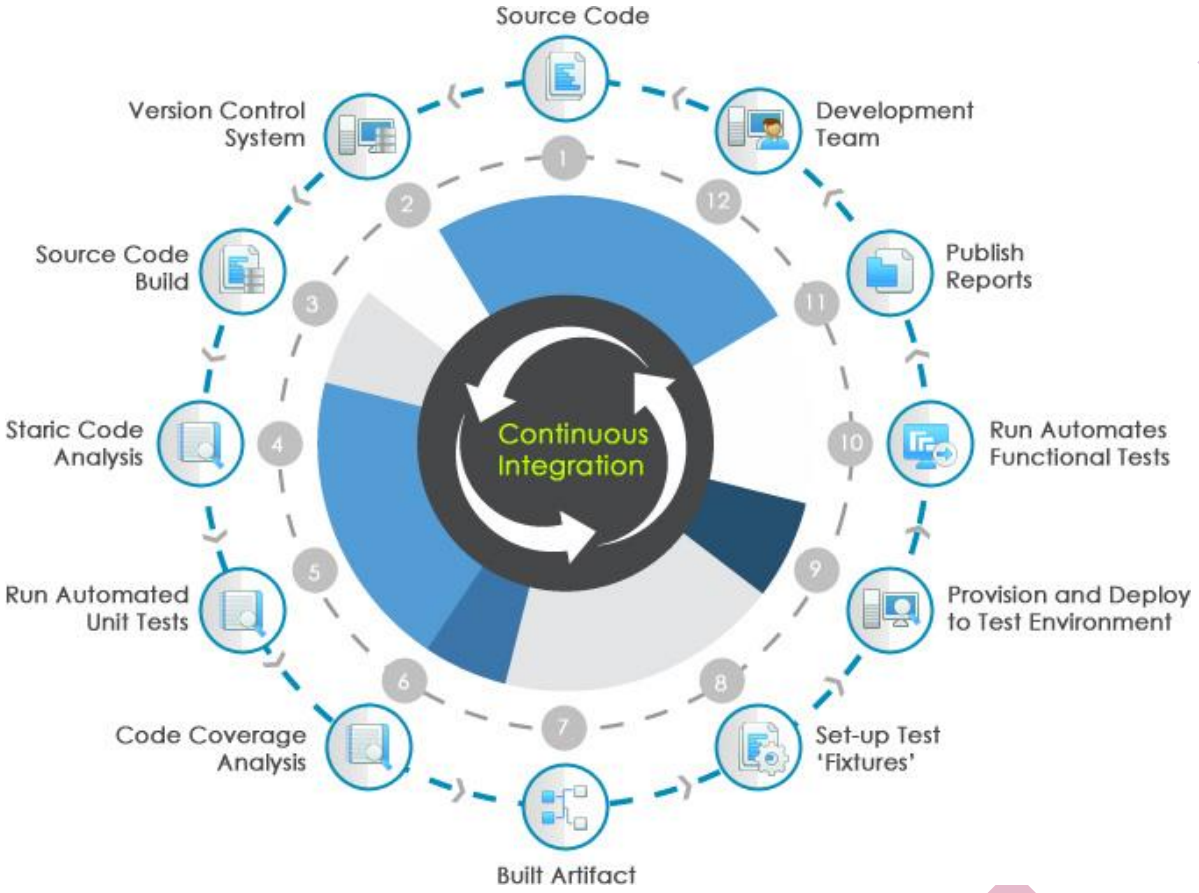
We will be starting soon!

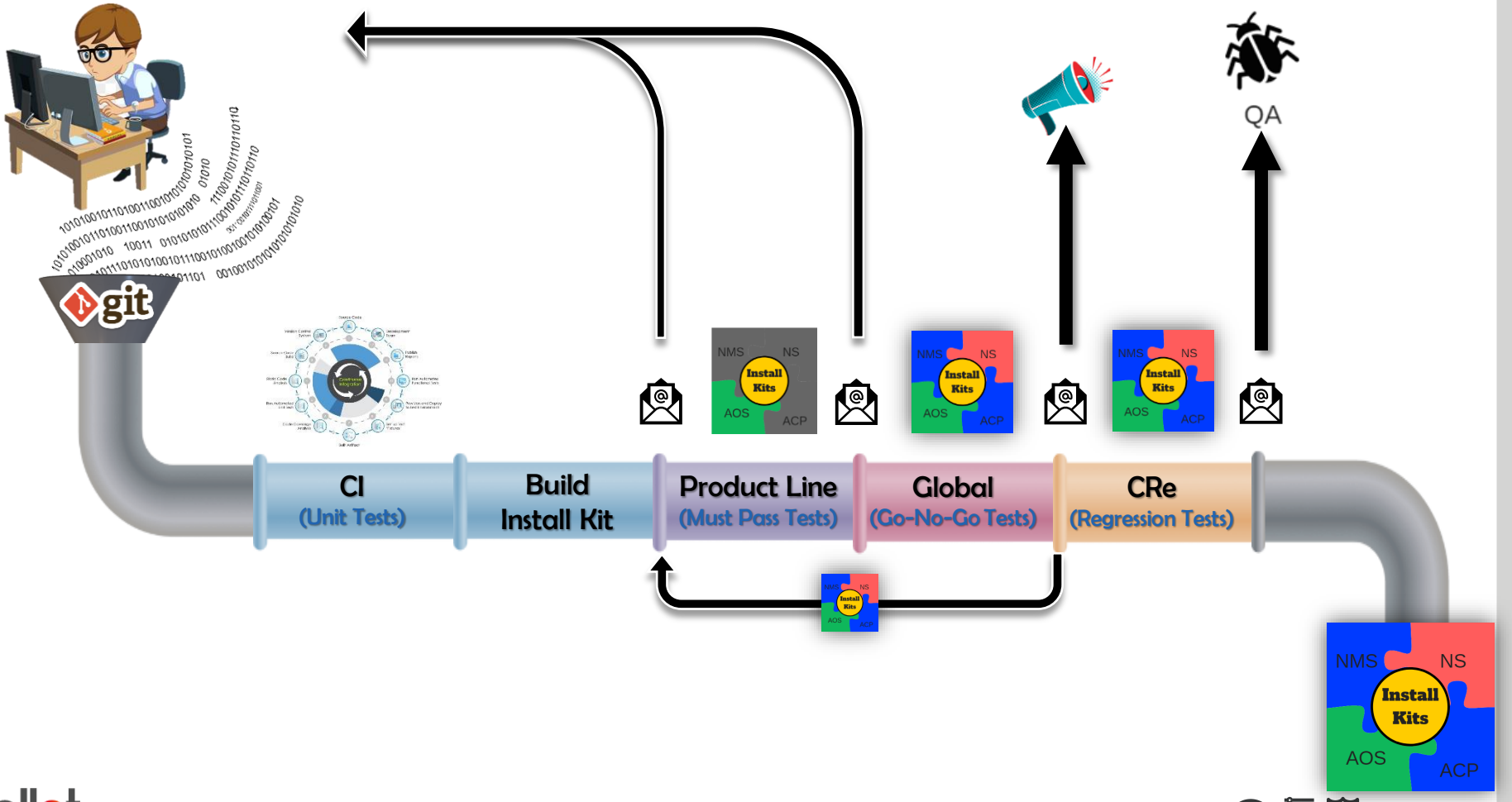
ribbon



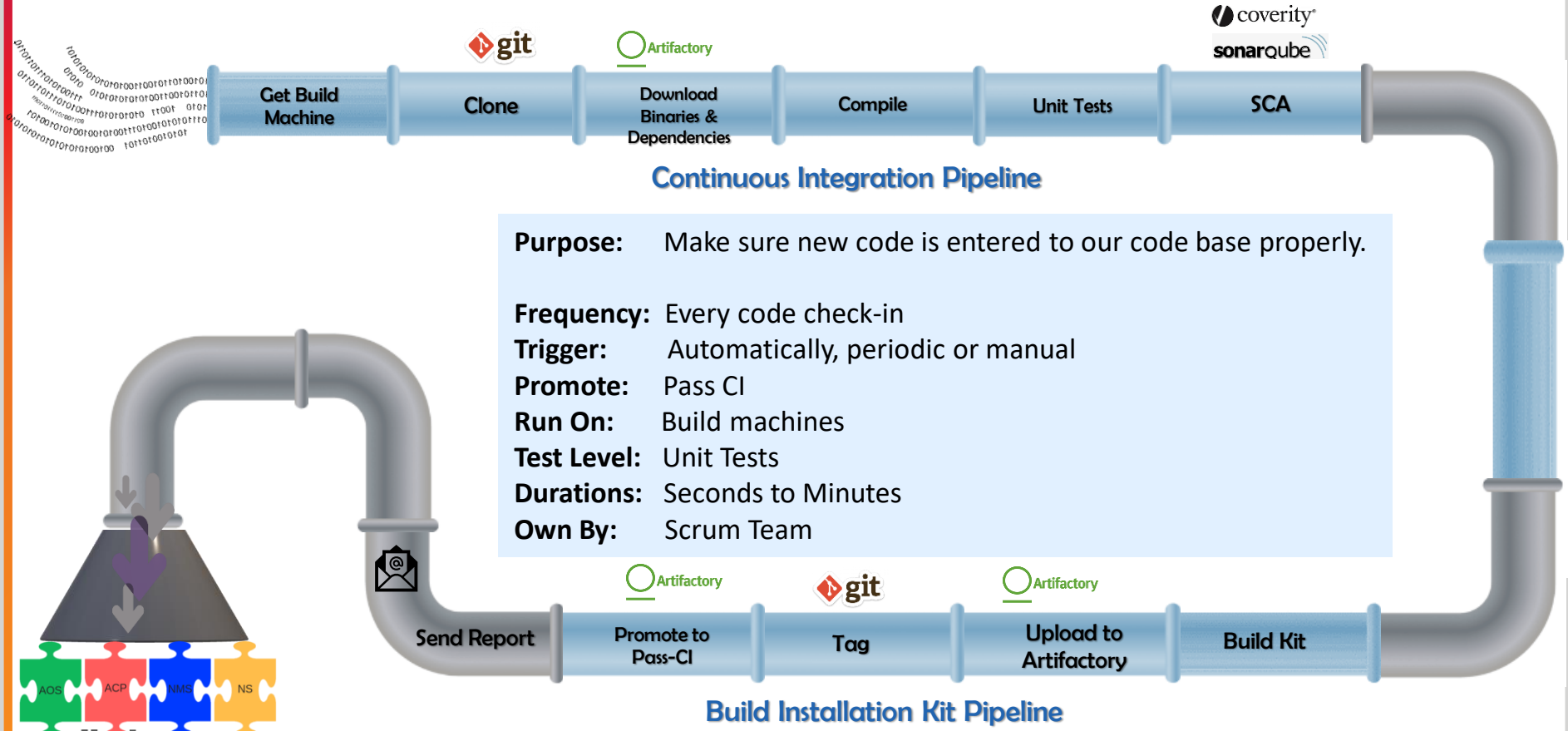
CI/CD

Continues Integration/Continues Delivery

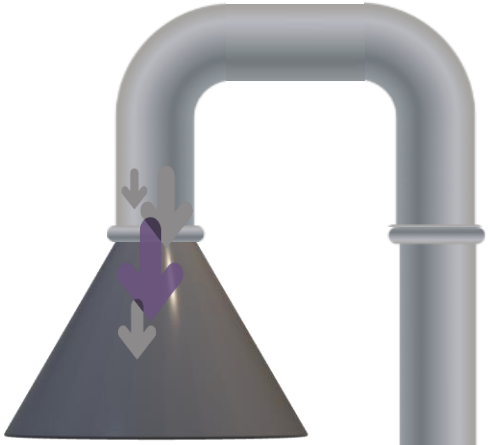

Source Code
Development Team
Publish Reports
Run Automates Functional Tests
Provision and Deploy to Test Environment
Set-up Test 'Fixtures'
Built Artifact
Code Coverage Analysis
Run Automated Unit Tests
Staric Code Analysis
Source Code Build
Version Control System
Continuous Integration
1
2
3
4
5
6
7
8
9
10
11
12


git
QA
NMS
NS
Install Kits
AOS
ACP
CI
(Unit Tests)
Build
Install Kit
Product Line
(Must Pass Tests)
Global
(Go-No-Go Tests)
CRe
(Regression Tests)

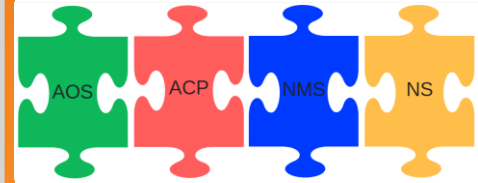
CI & Build Kit pipelines



Product Line Pipeline

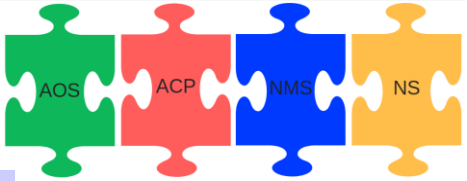


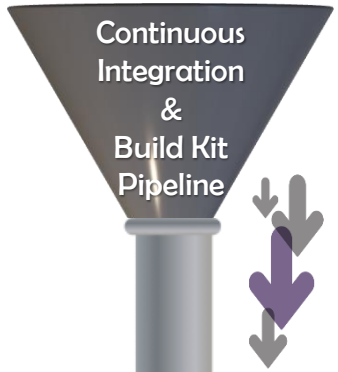
Product Under Tests
(Functionality & Integration)
Pass!!!



Purpose: Make sure our product didn't break the basic functionality and Integration with other products

Frequency: Up-to 4 times a day
Trigger: Periodic or manual
Promote: Pass PL
Run On: Small virtual environments (base on SGVE)
Test Level: Must Pass
Durations: Up to 120 Minutes
Own By: Scrum Team



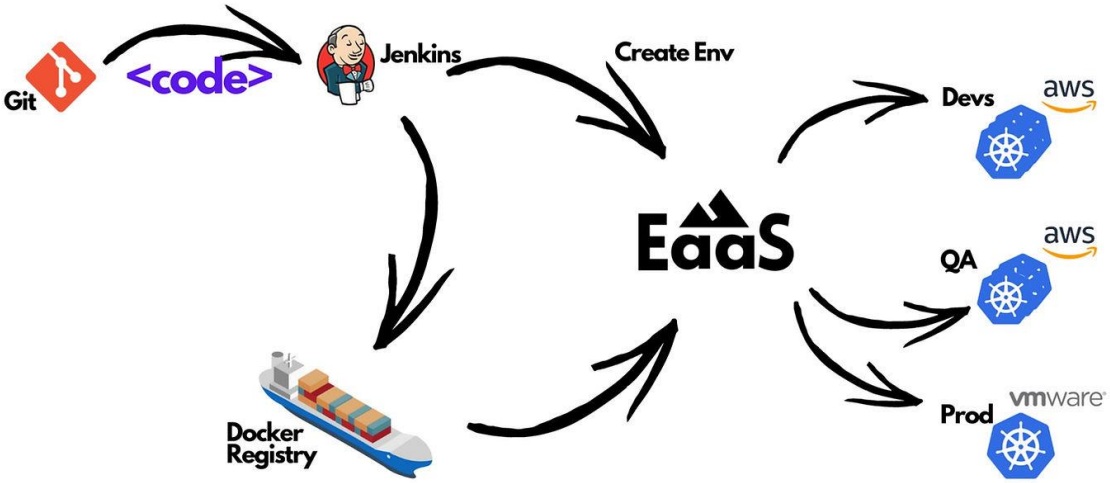


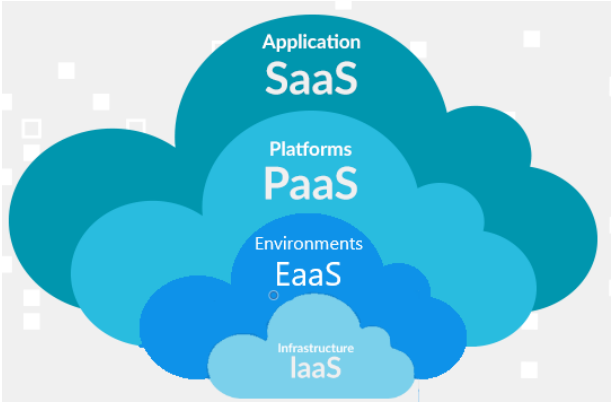
Global Pipeline

Purpose: Making sure all products work properly together. Installing all latest kits from scratch on an environment received as a service.
+ Create new fixture for next builds

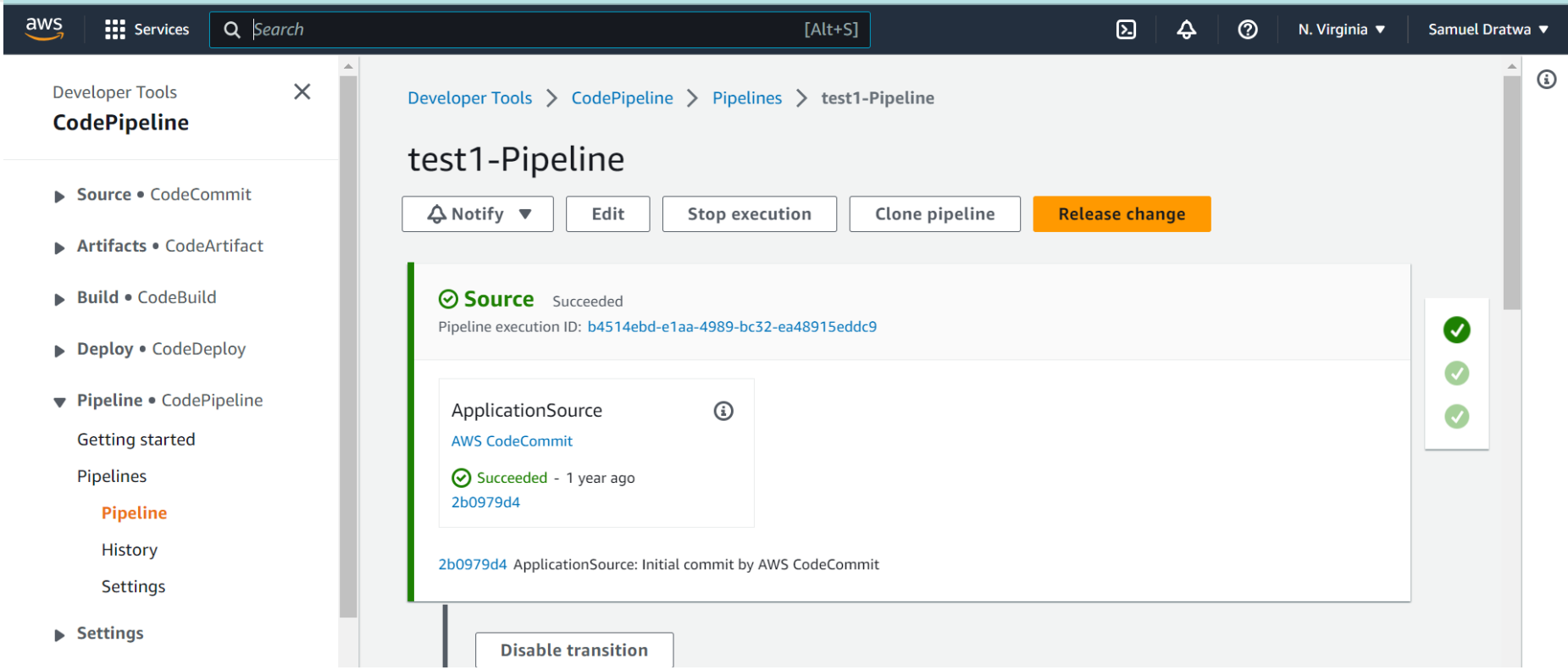
Frequency: Once a day
Trigger: Periodic or manual
Promote: Pass Global
Run On: Environments base on SGVE, SGT & SG9K
Test Level: Go-No-Go
Durations: up to 180 Minutes
Own By: CI Team

Environment as a Service





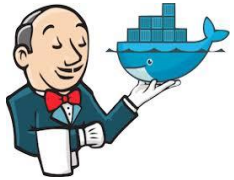
AWS Pipeline

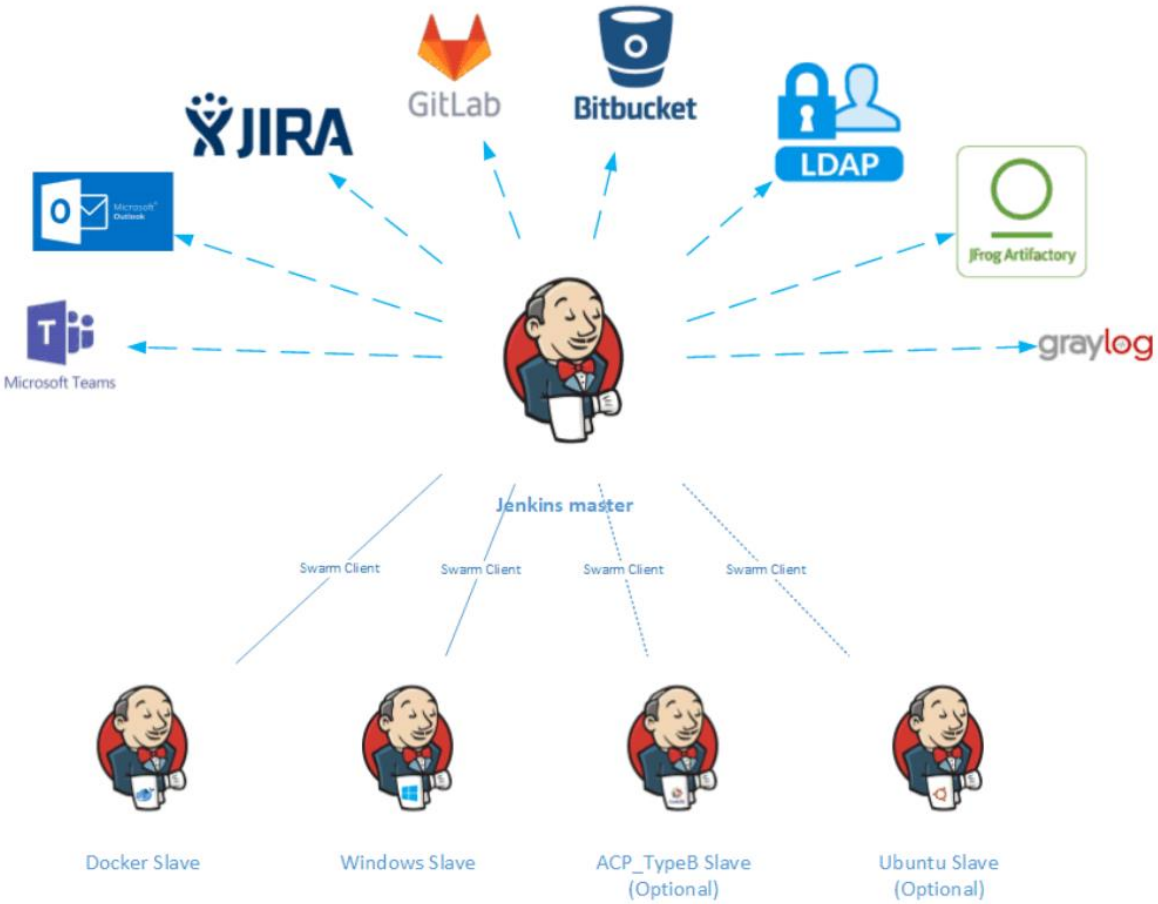


CI-CD Tech Stack

Jenkins

The leading open-source automation server, Jenkins provides hundreds of plugins to support building, deploying and automating any project.

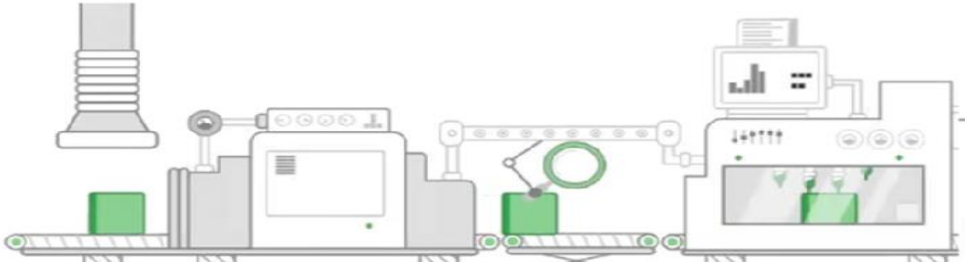



JIRA
GitLab
Bitbucket
LDAP
Microsoft Teams
JFrog Artifactory
graylog
Jenkins master
Swarm Client
Swarm Client
Swarm Client
Swarm Client
Docker Slave
Windows Slave
ACP_TypeB Slave
(Optional)
Ubuntu Slave
(Optional)

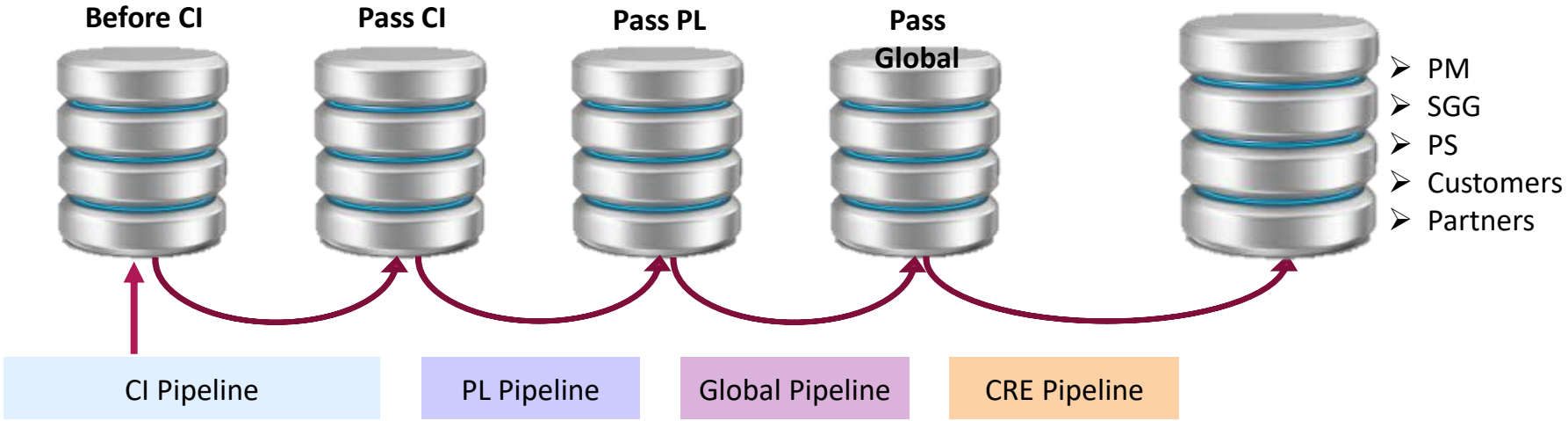
Artifactory

A binary repository manager which designed to optimize the download and storage of binary files.

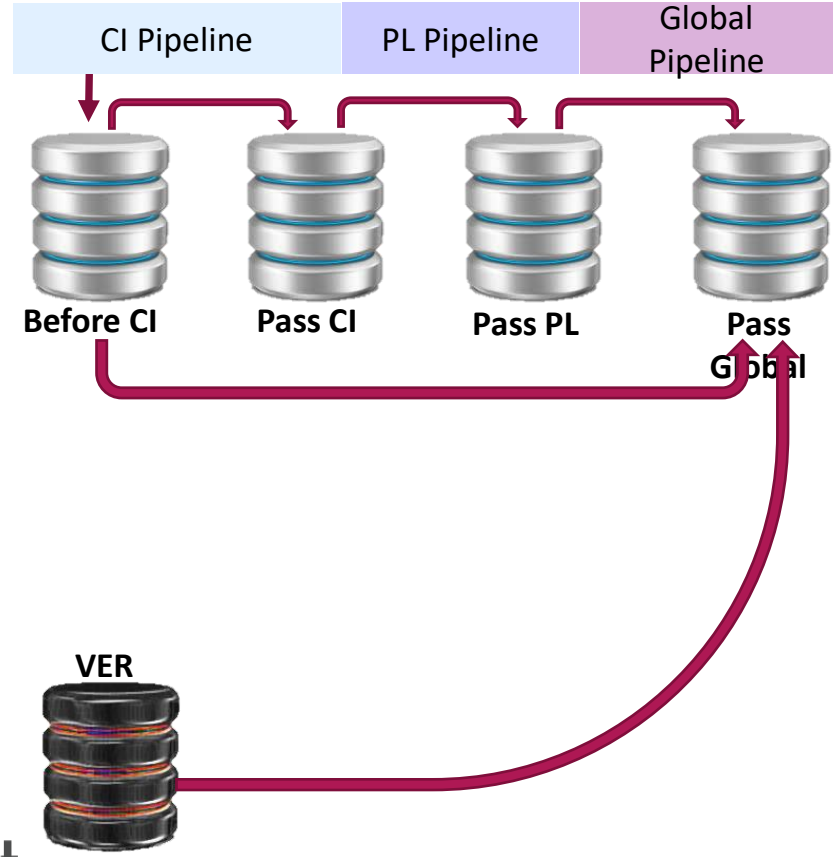
JFrog Artifactory



Artifactory Repositories



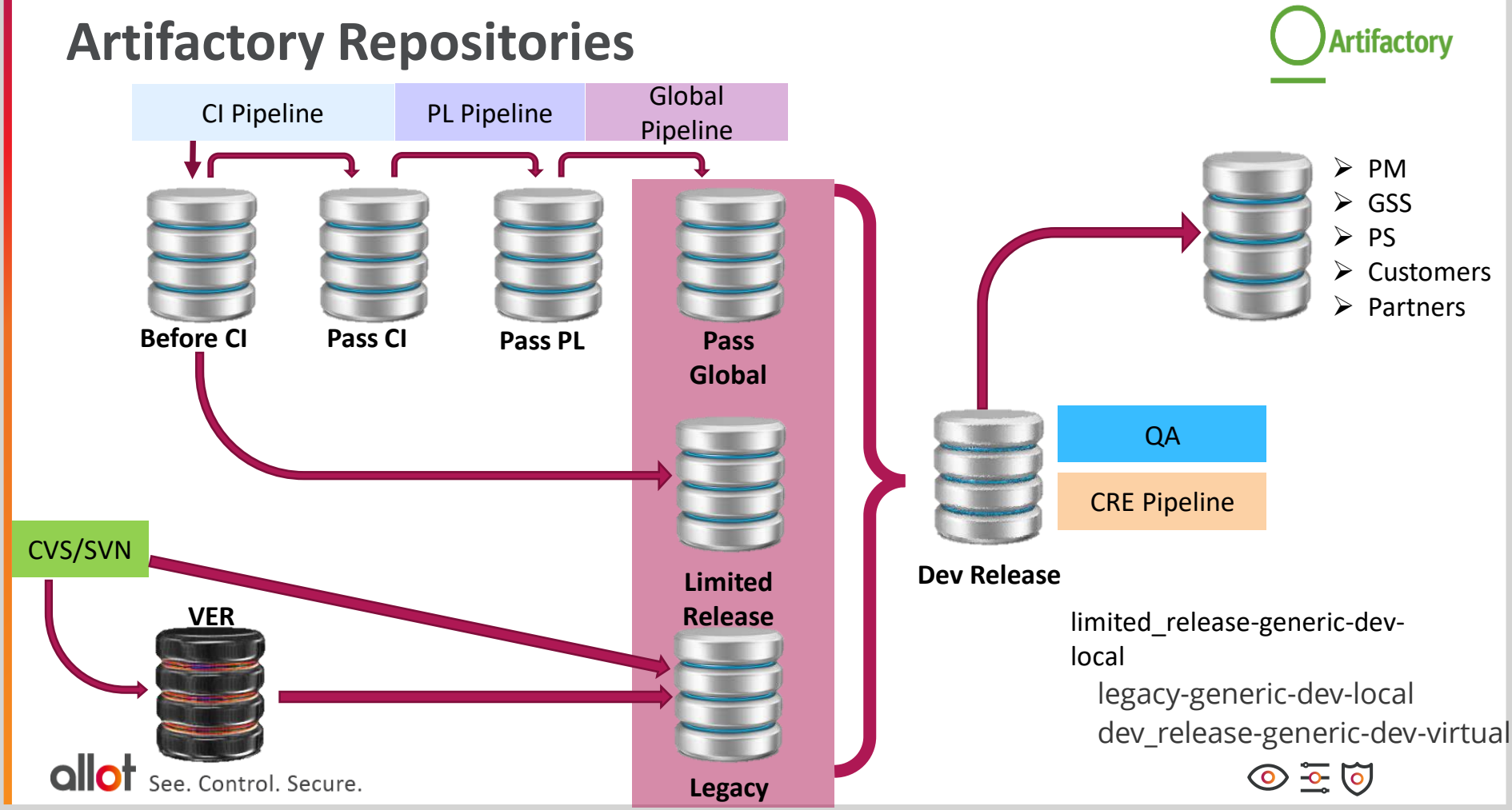
Artifactory Repositories



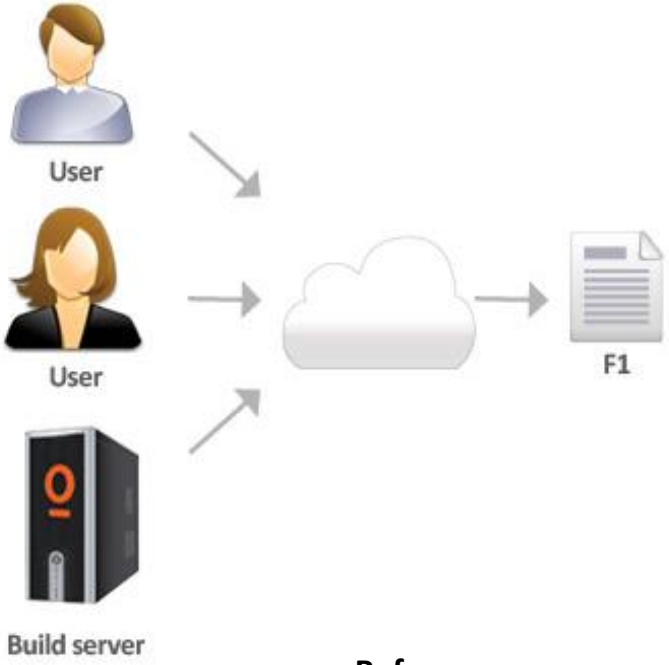
QA

CRE Pipeline

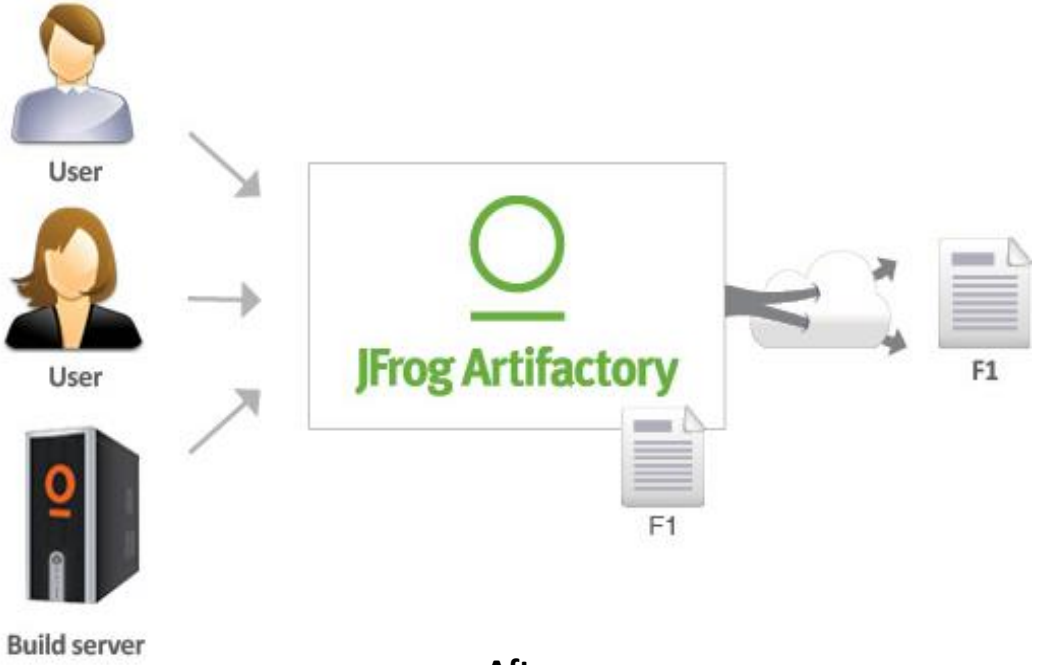
Artifactory Repositories



Artifactory as a gateway



Before



After

Docker Containers as a Service Platform

Build → **Ship** → **Run**

Docker Toolbox

- “docker push” with image signing

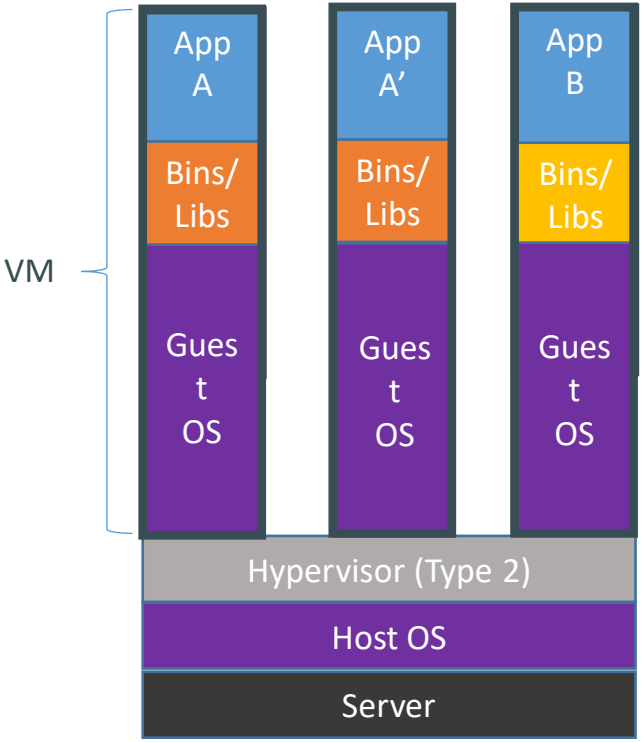
Docker Trusted Registry

- Search/browse repos
- Teams-based RBAC
- View signed images
- Deleting tags

Docker Universal Control Plane

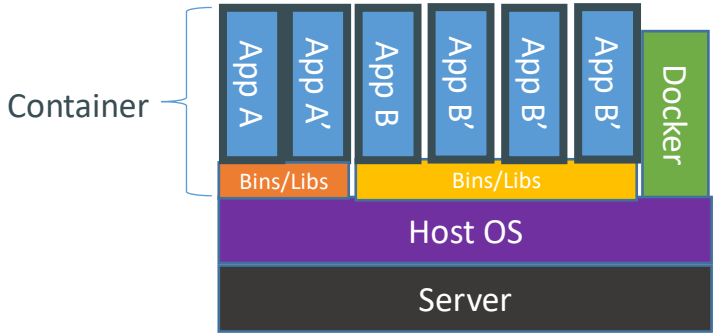
- Authentication
- Deploy and scale-out app
- Monitor stats
- Secrets management

Containers vs. VMs



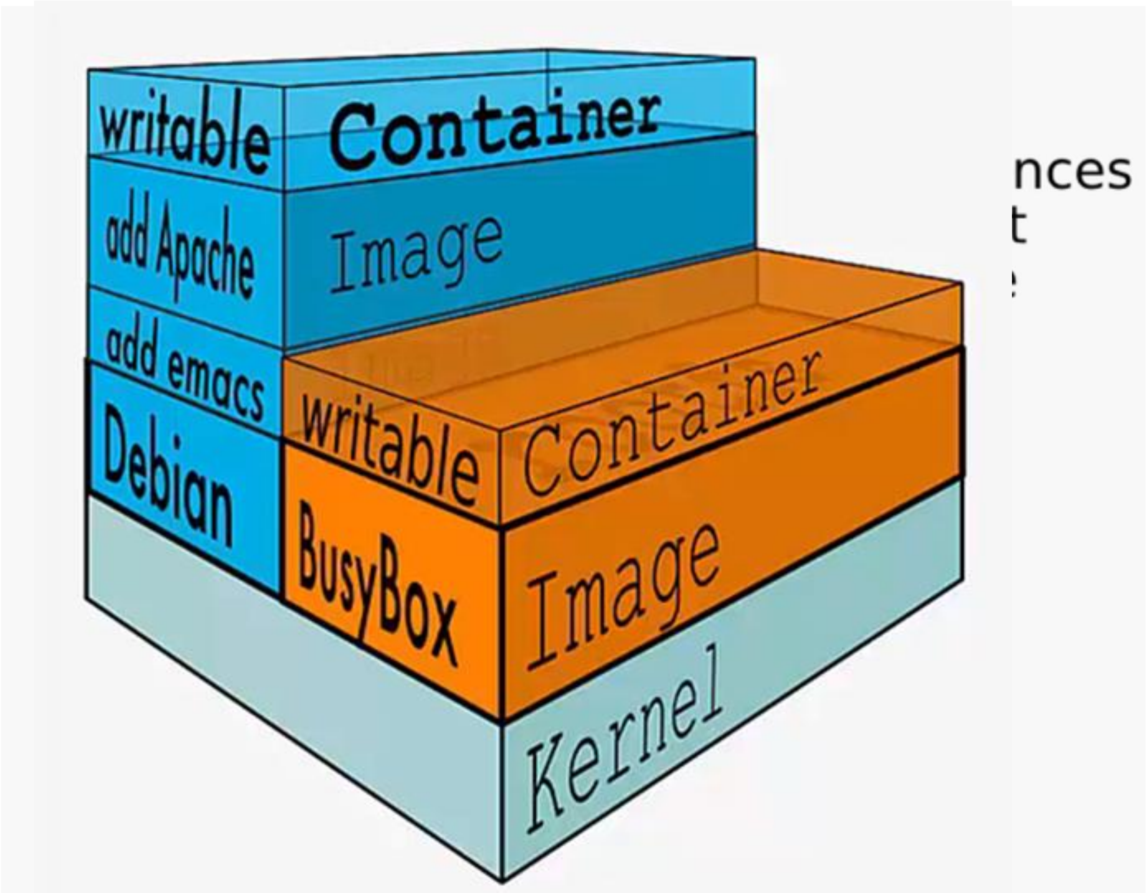
Containers are isolated, but share OS and, where appropriate, bins/libraries

…result is significantly faster deployment, much less overhead, easier migration, faster restart

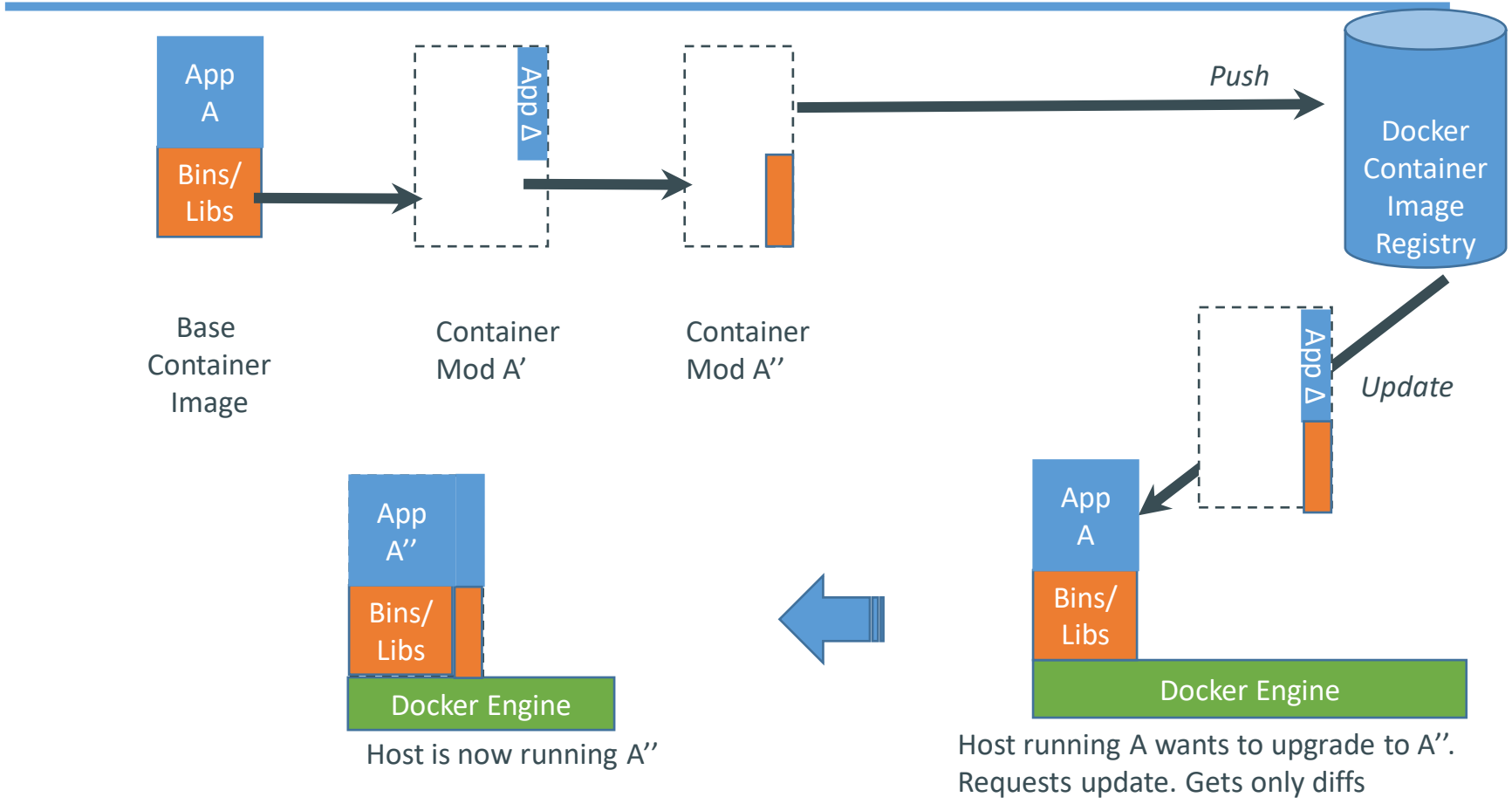


docker

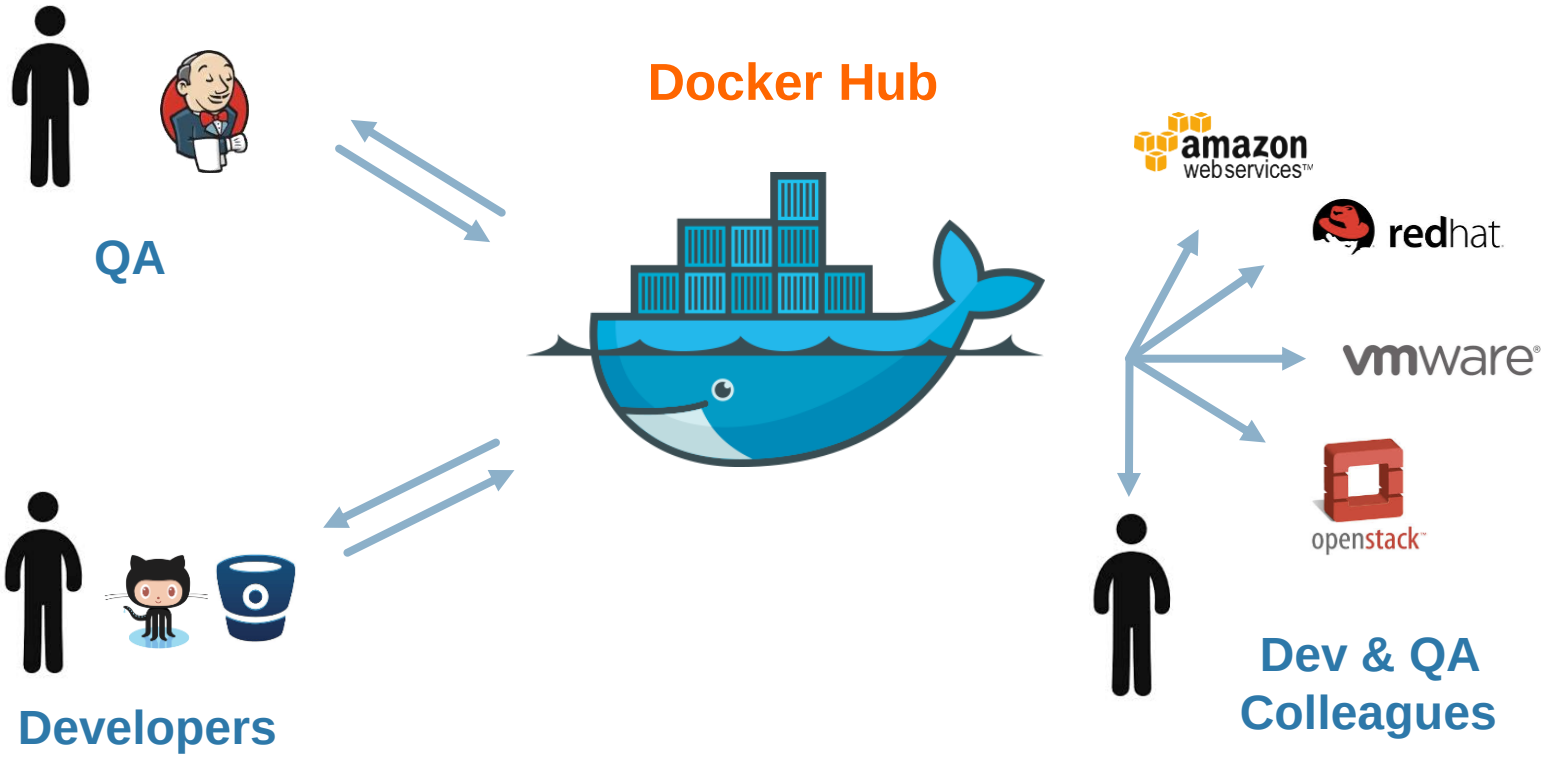
Image layers



Changes and Updates

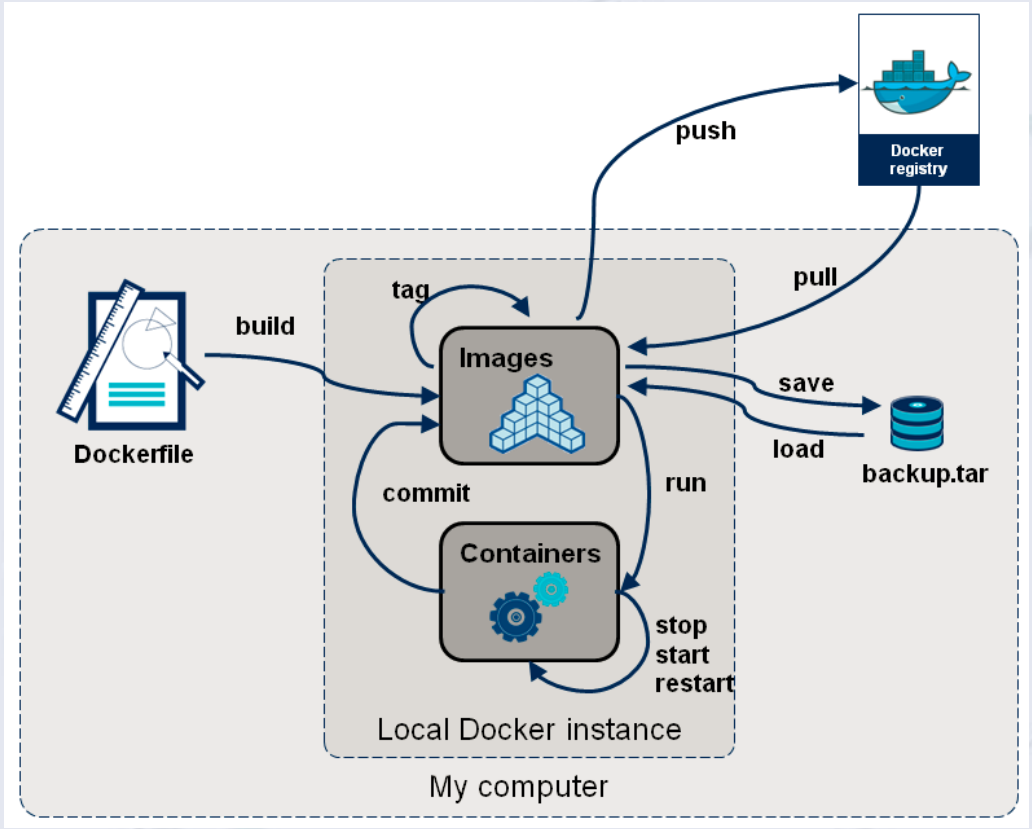


Build & Ship

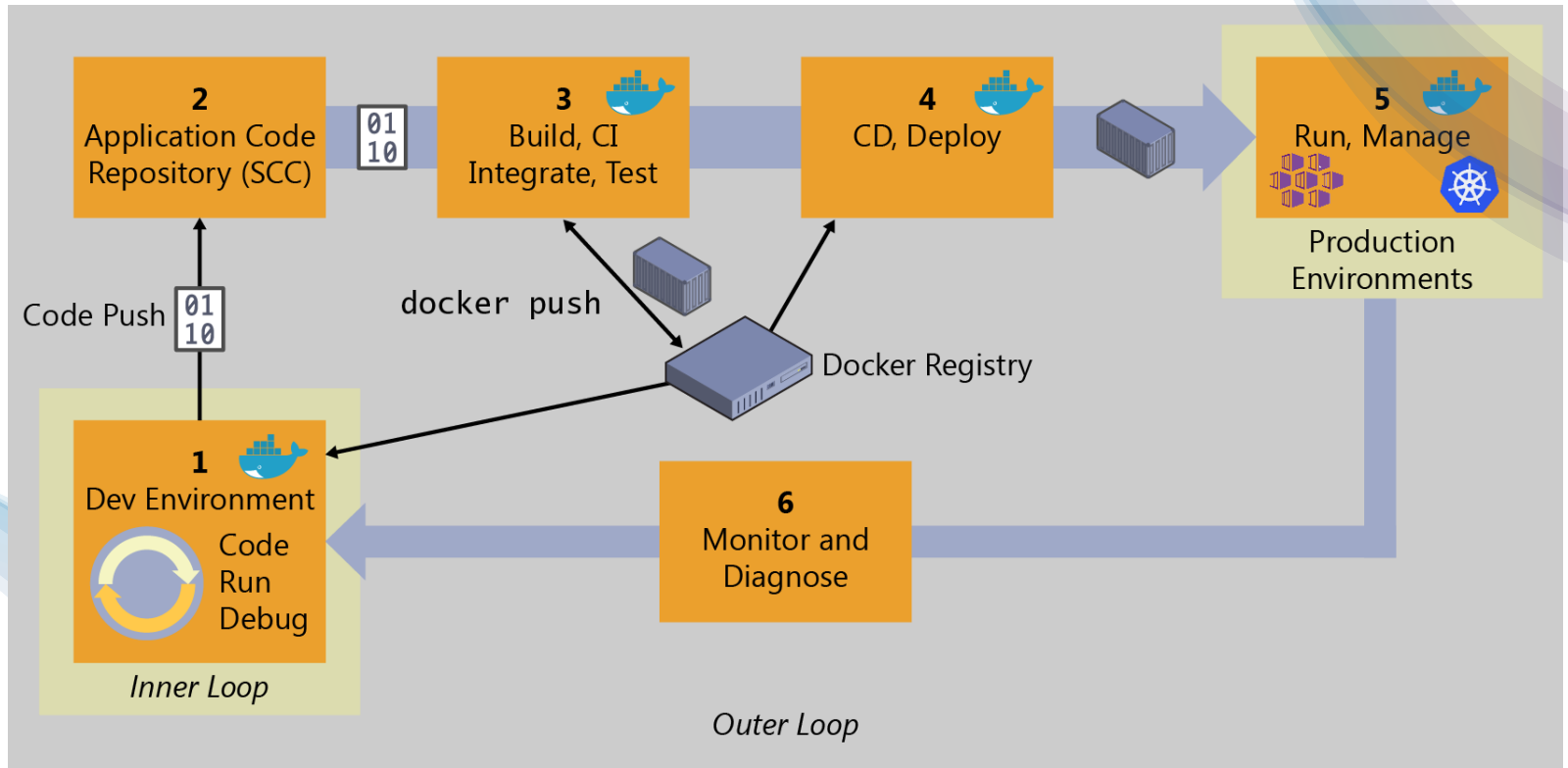

Docker Hub
QA
Developers
Dev & QA
Colleagues
amazon
web services
redhat
vmware
openstack

Docker Container Lifecycle

- The Life of a Container
  - Conception
    - **BUILD** an Image from a Dockerfile
  - Birth
    - **RUN** (create+start) a container
  - Reproduction
    - **COMMIT** (persist) a container to a new image
    - **RUN** a new container from an image
  - Sleep
    - **KILL/stop** a running container
  - Wake
    - **START** a stopped container
  - Death
    - **RM** (delete) a stopped container
- Extinction
  - **RMI** a container image (delete image)



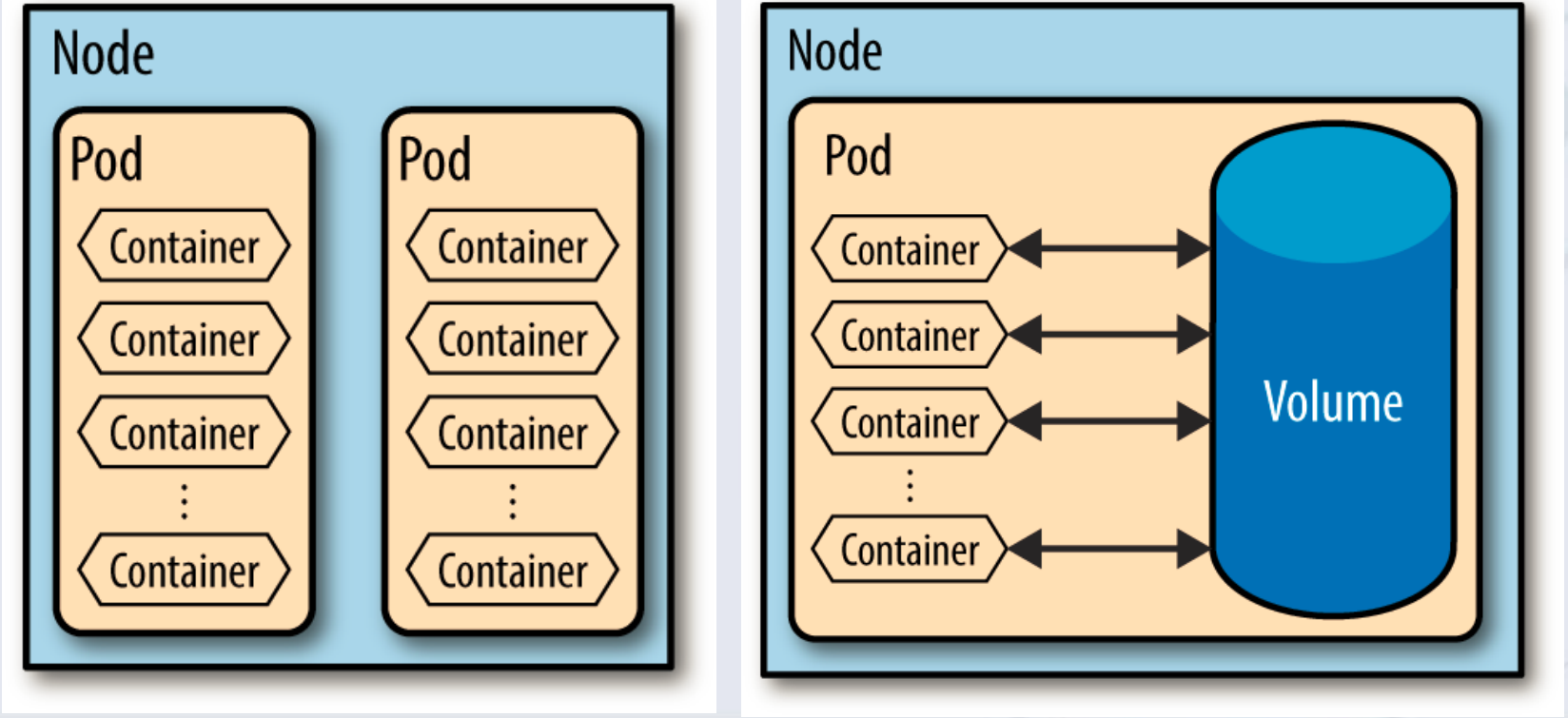
Docker application life cycle workflow



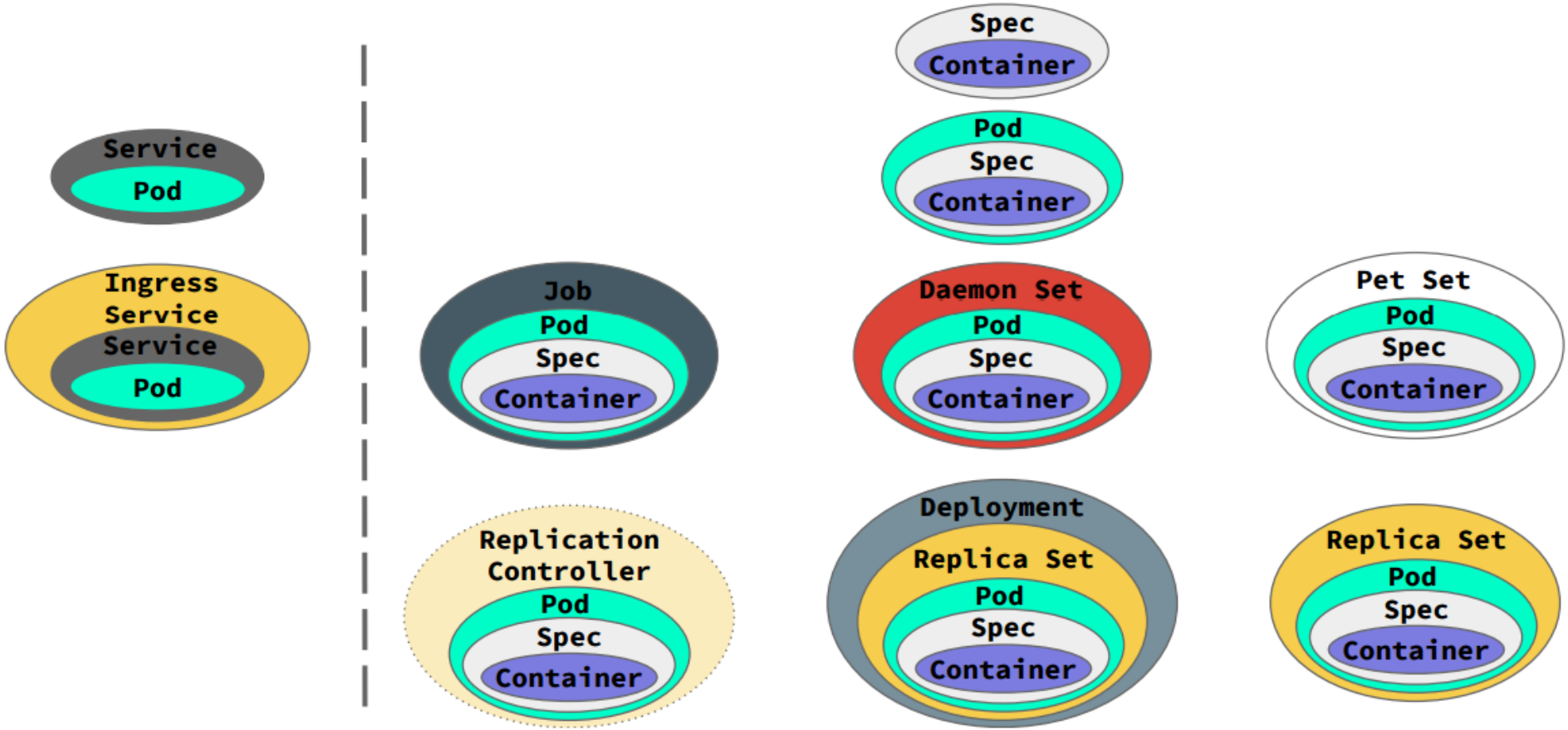
- The inner loop consists of typical steps like "code," "run," "test," and "debug," plus the additional steps needed right before running the app locally. This is the developer's process to run and test the app as a Docker container. The inner-loop workflow will be explained in the sections that follow.

Pods

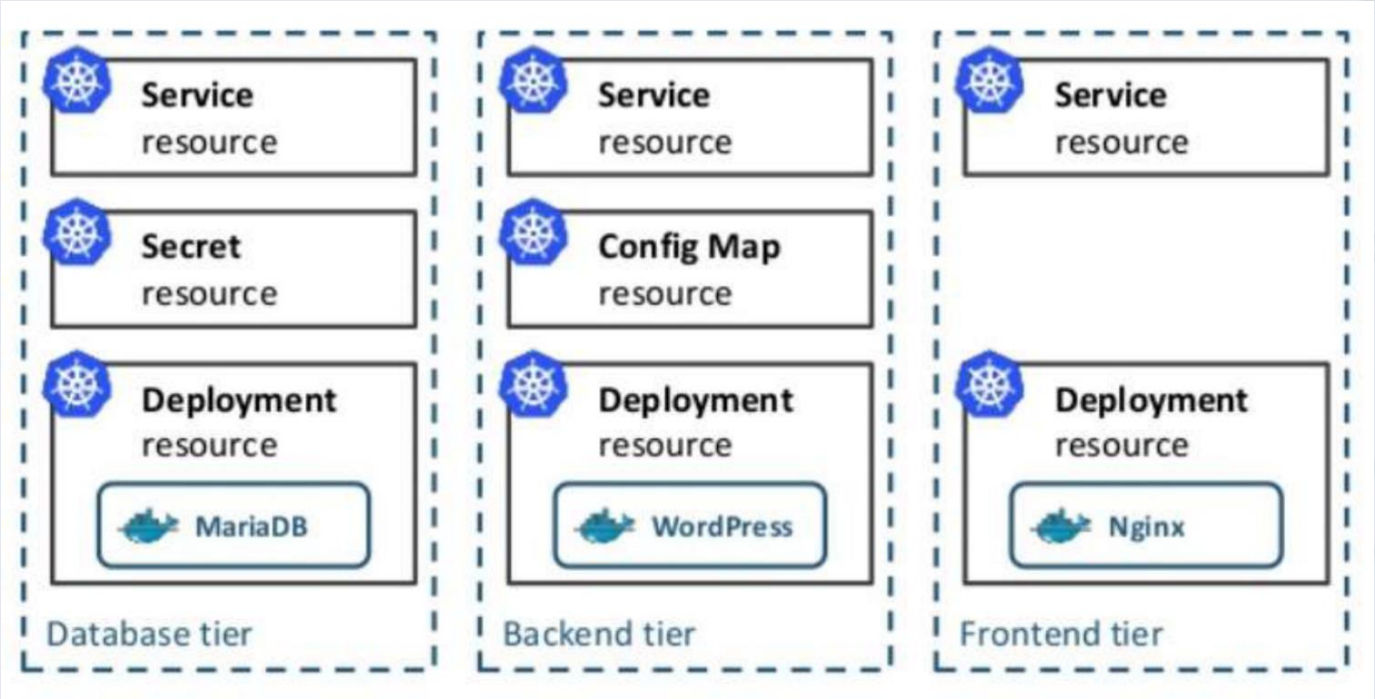
A *pod* is a collection of containers and volumes that are bundled and scheduled together because they share a common resource, usually **a filesystem** or **IP address**.



Pod types / kind



Kubernetes Pain Point

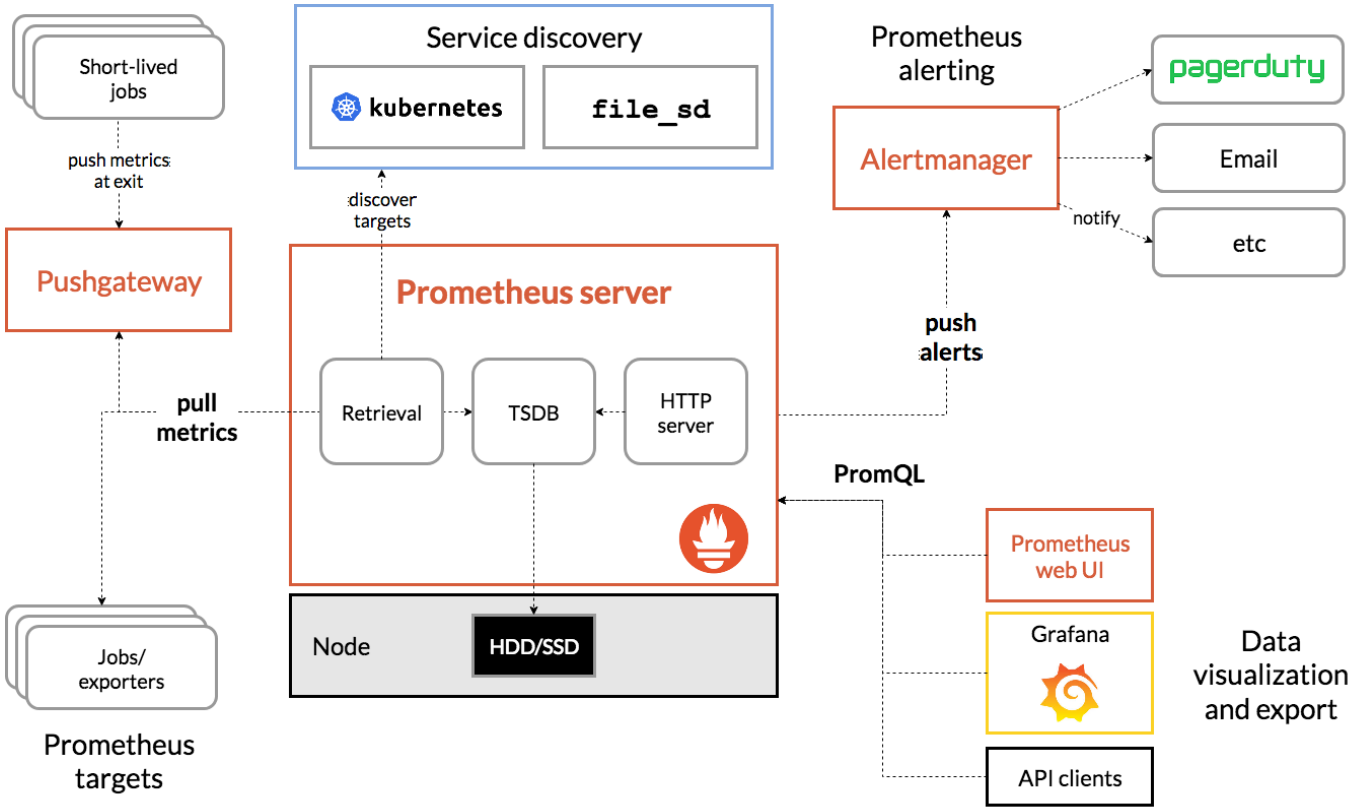


- 1 MicroService == 1 POD + 1 Deployment + 1 ReplicationSet + 1 Service (+ 1 Ingress)
- Lifecycle Management of Applications is **challenging**

What is Helm

- Package manager for K8s
- like yum, apt, pip but for Kubernetes Application
- Search and reuse or start from scratch
- Lifecycle Management
  - Create
  - Install
  - Upgrade/Rollback
  - Delete
  - Status
  - Versioning

Prometheus structure



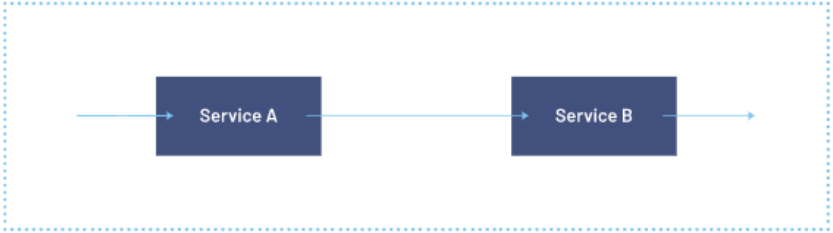
Metrics format

- "Help" string - describes the metric
- "Type" string - has 4 possible values:
  - *Counter* - counts occurrences (value can only increase)
  - *Gauge* - informs current value (value can be increased/decreased)
  - *Histogram* - Measures the frequency of values occurrences within a range by range-buckets
  - *Summary* - Captures individual observations from an event and summarizes them into several related metrics:
    - Sum of all values (aka observations)
    - Observation count
    - List of quantiles to compute (e.g. 0.1, 0.5 (median), 0.75, 0.95)

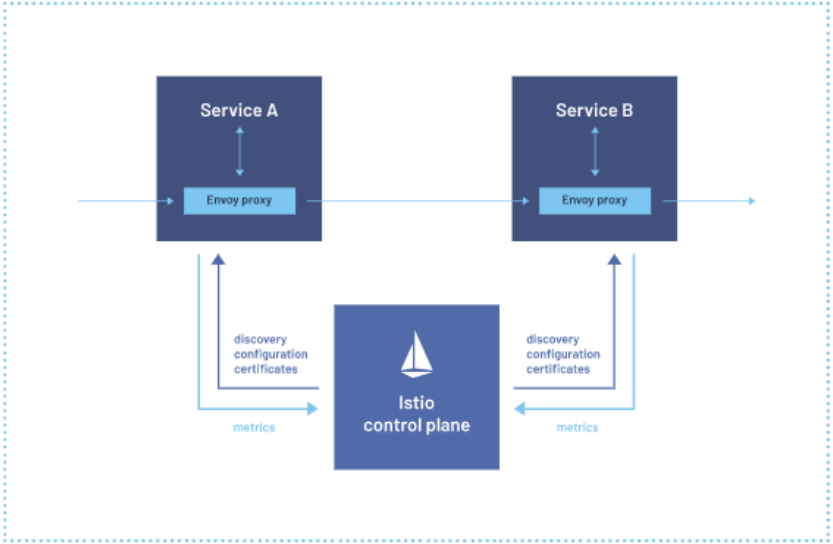
```
# HELP http_requests_total The total number of HTTP requests.
# TYPE http_requests_total counter
# This is a comment
http_requests_total{method="post",code="200"} 1027 1395066363000
http_requests_total{method="post",code="400"}    3 1395066363000
```

Istio

An open platform to connect, manage, and secure microservices



Before utilizing Istio

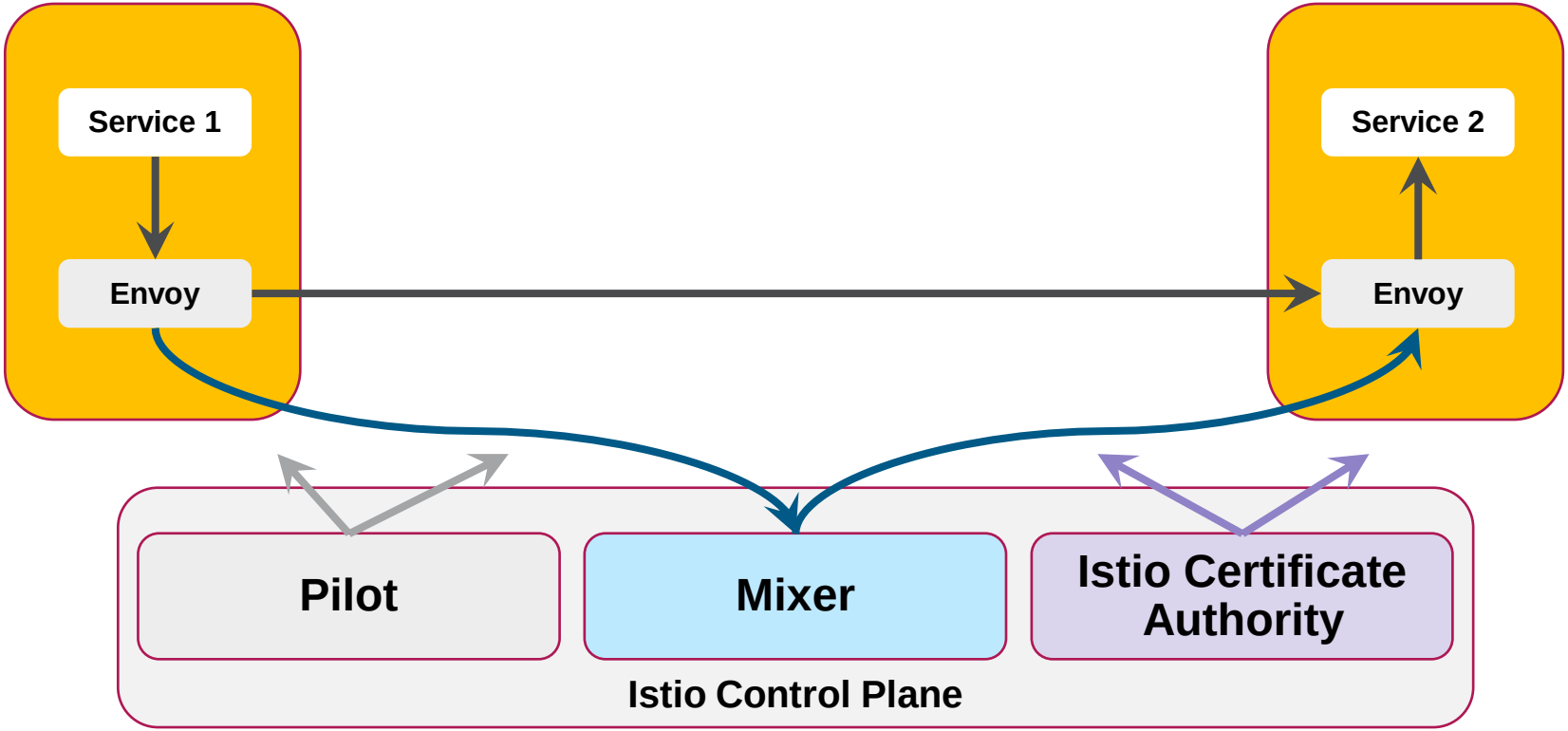


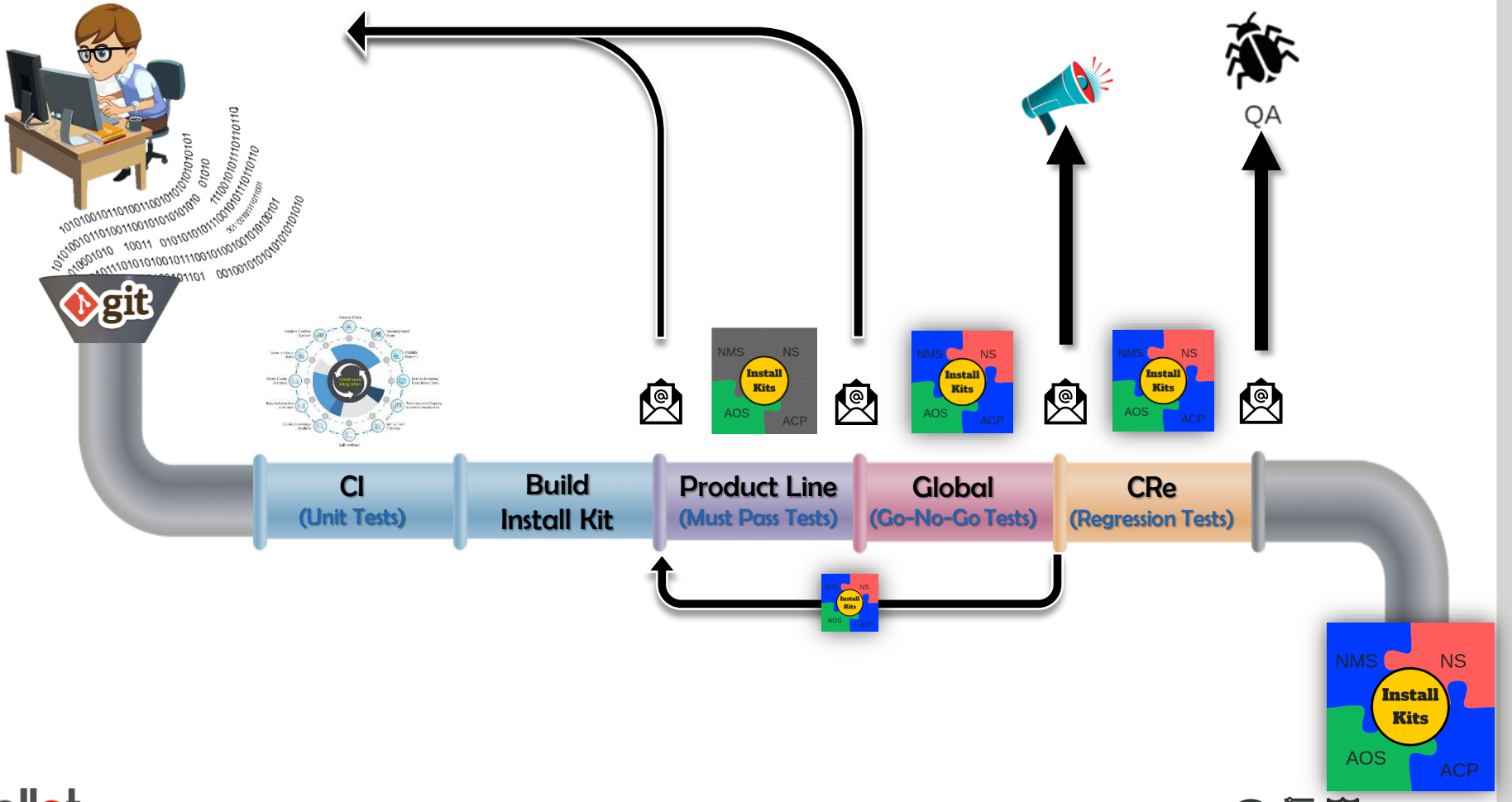
After utilizing Istio

Istio is an open source service mesh that layers transparently onto existing distributed applications. Istio's powerful features provide a uniform and more efficient way to secure, connect, and monitor services. Istio is the path to load balancing, service-to-service authentication, and monitoring – with few or no service code changes. Its powerful control plane brings vital features, including:

- Secure service-to-service communication in a cluster with TLS encryption, strong identity-based authentication and authorization
- Automatic load balancing for HTTP, gRPC, WebSocket, and TCP traffic
- Fine-grained control of traffic behavior with rich routing rules, retries, failovers, and fault injection
- A pluggable policy layer and configuration API supporting access controls, rate limits and quotas
- Automatic metrics, logs, and traces for all traffic within a cluster, including cluster ingress and egress

Istio Control Plane




git
QA
NMS
NS
Install Kits
AOS
ACP
CI
(Unit Tests)
Build
Install Kit
Product Line
(Must Pass Tests)
Global
(Go-No-Go Tests)
CRe
(Regression Tests)

Google best practices recommendation

The following table summarizes the tasks that we recommend when you use a CI/CD pipeline in GKE:

Area	Tasks
Continuous integration	☐ Create pipelines that enable rapid iteration. ☐ Follow the best practices for building containers. ☐ Test your container images. ☐ Establish security early in pipelines.
Continuous delivery	☐ Use GitOps methodology. ☐ Promote, rather than rebuild containers. ☐ Consider using more advanced deployment and testing patterns. ☐ Separate clusters for different environments. ☐ Keep pre-production environments close to production. ☐ Prepare for failures in production.