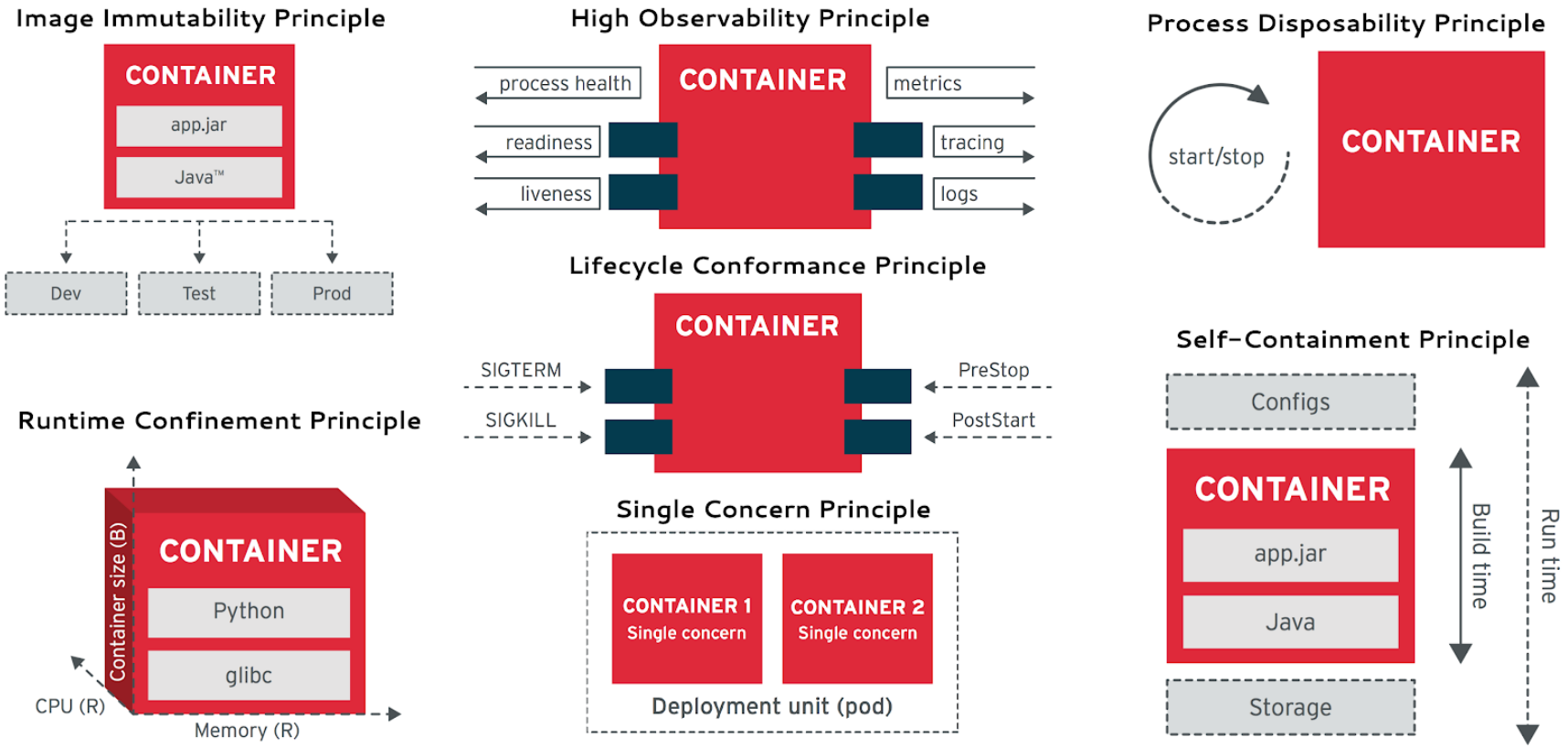
Kubernetes

Agenda

- Introduction
- Kubernetes Architecture
- Cluster
- Resources / Objects
- HPA
- Namespace
- Helm

Introduction

Principles of Container-based



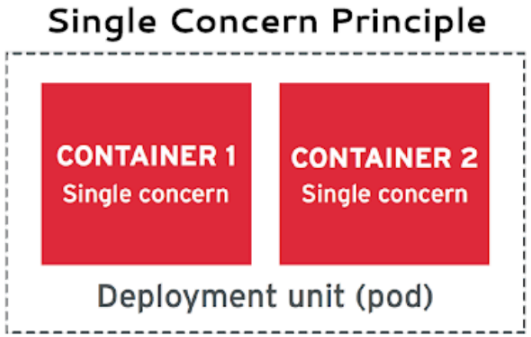
Principles of Container-based

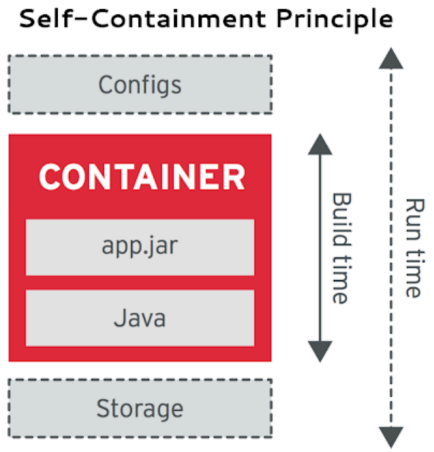
Build Time

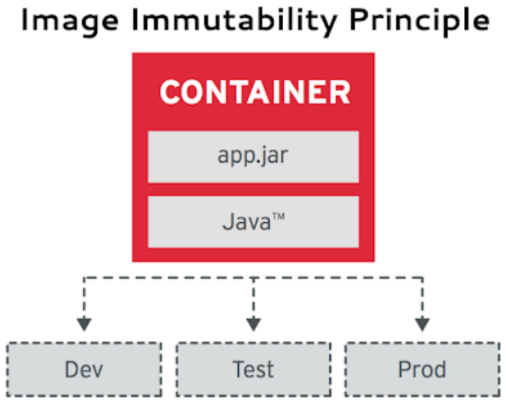
Single Concern: Each container addresses a single concern and does it well.

Self-Containment: A container relies only on the presence of the Linux kernel. Additional libraries are added when the container is built.

Image Immutability: Containerized applications are meant to be immutable, and once built are not expected to change between different environments.







Principles of Container-based

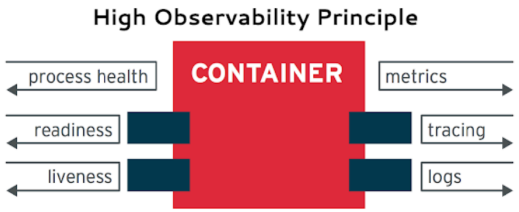
Run Time

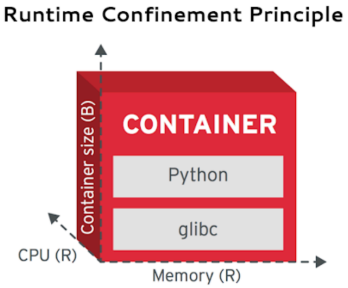
High Observability: Every container must implement all necessary APIs to help the platform observe and manage the application in the best way possible.

Lifecycle Conformance: A container must have a way to read events coming from the platform and conform by reacting to those events.

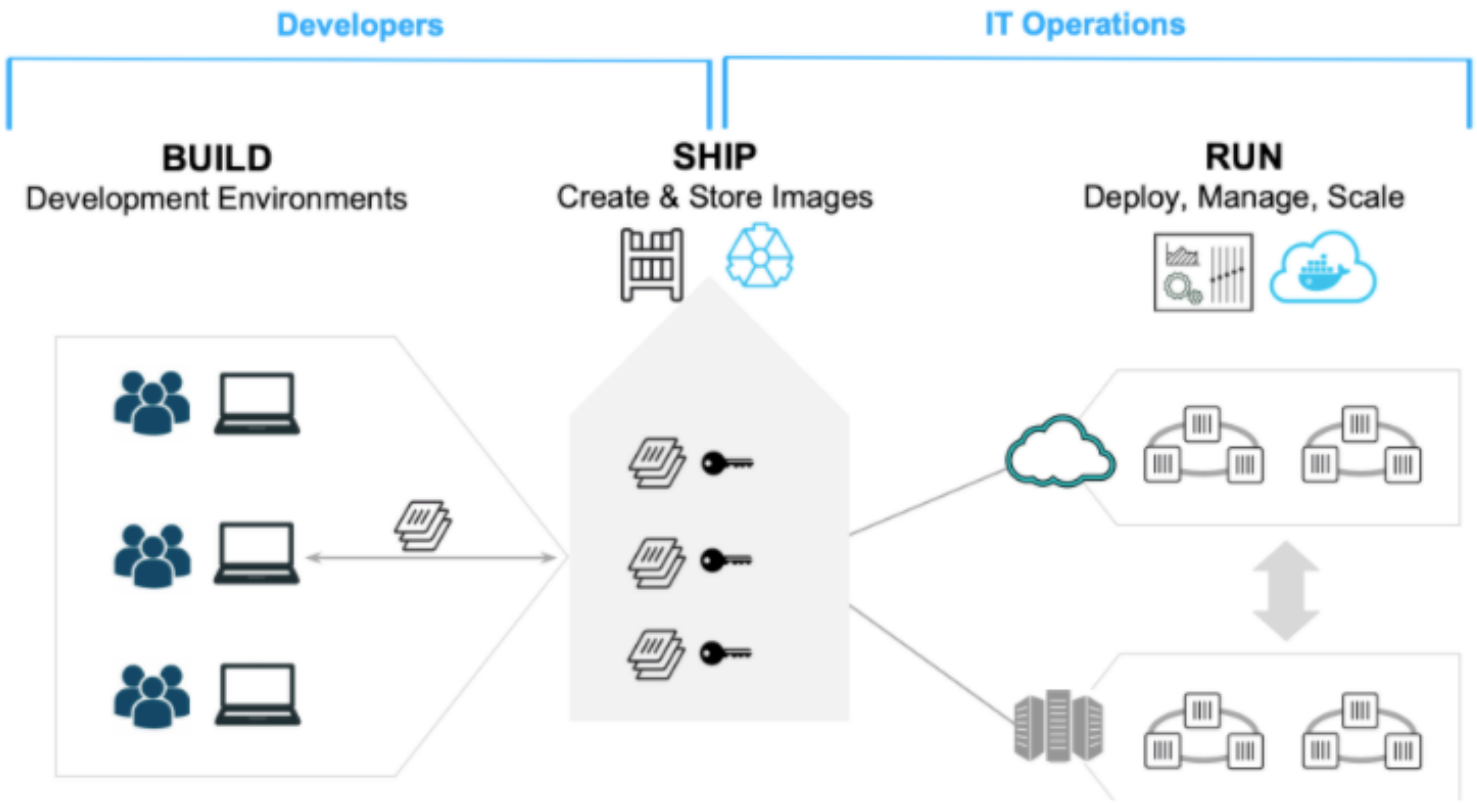
Process Disposability: Containerized applications must be as ephemeral as possible and ready to be replaced by another container instance at any point in time.

Runtime Confinement: Every container must declare its resource requirements and restrict resource use to the requirements indicated.

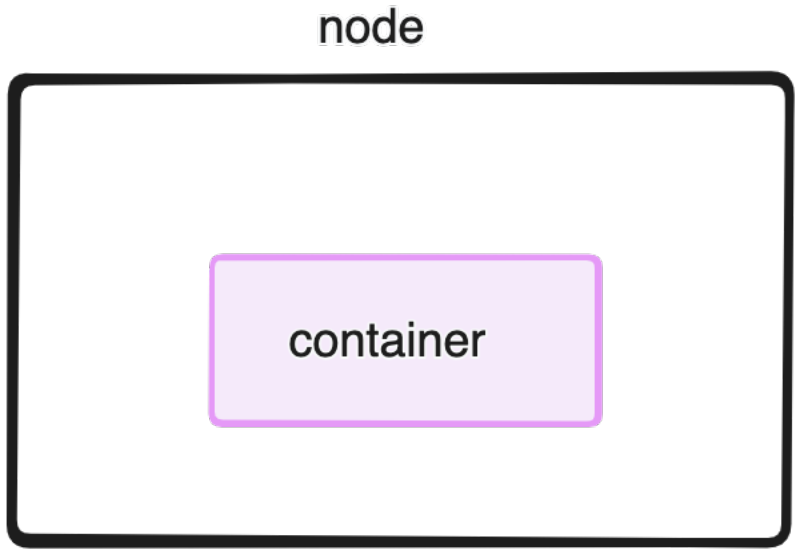




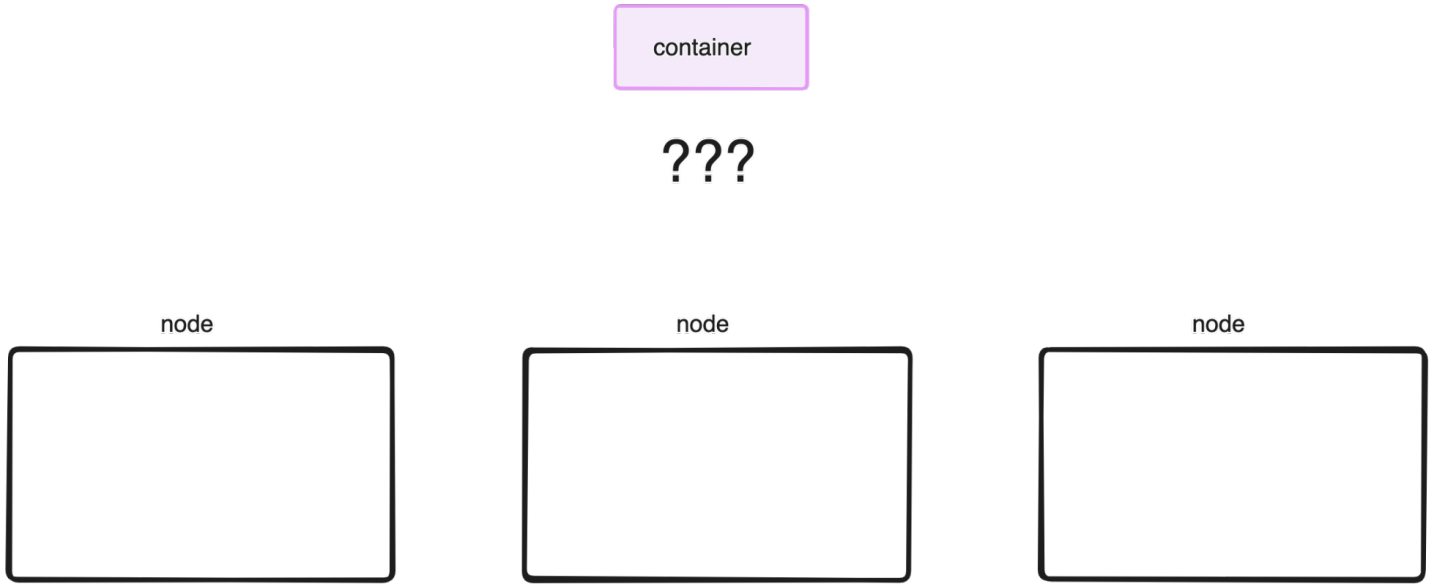
Using Docker: Build, Ship, Run



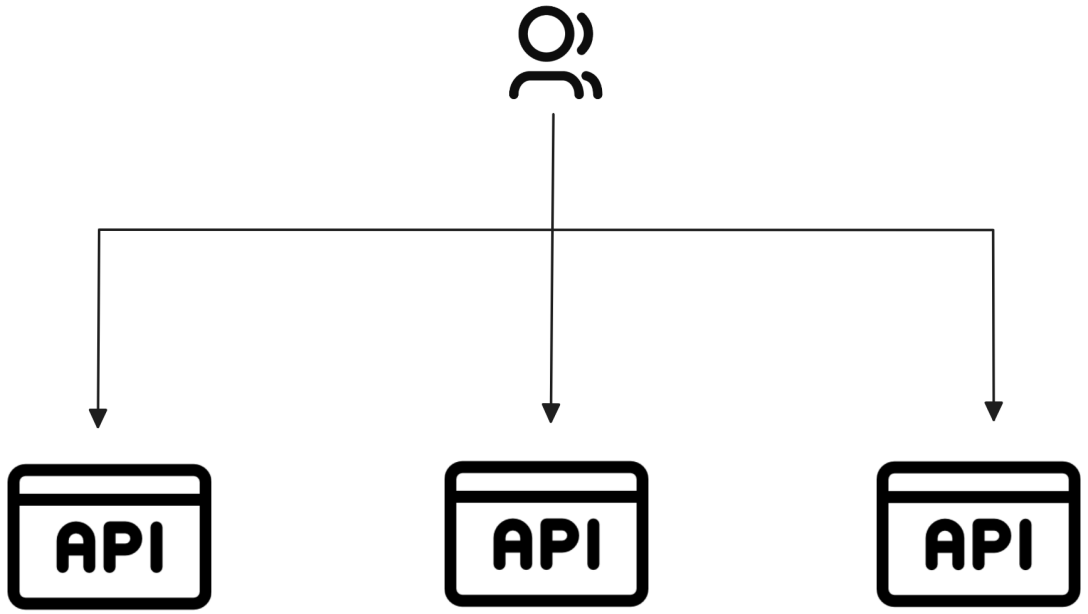
Docker standalone server



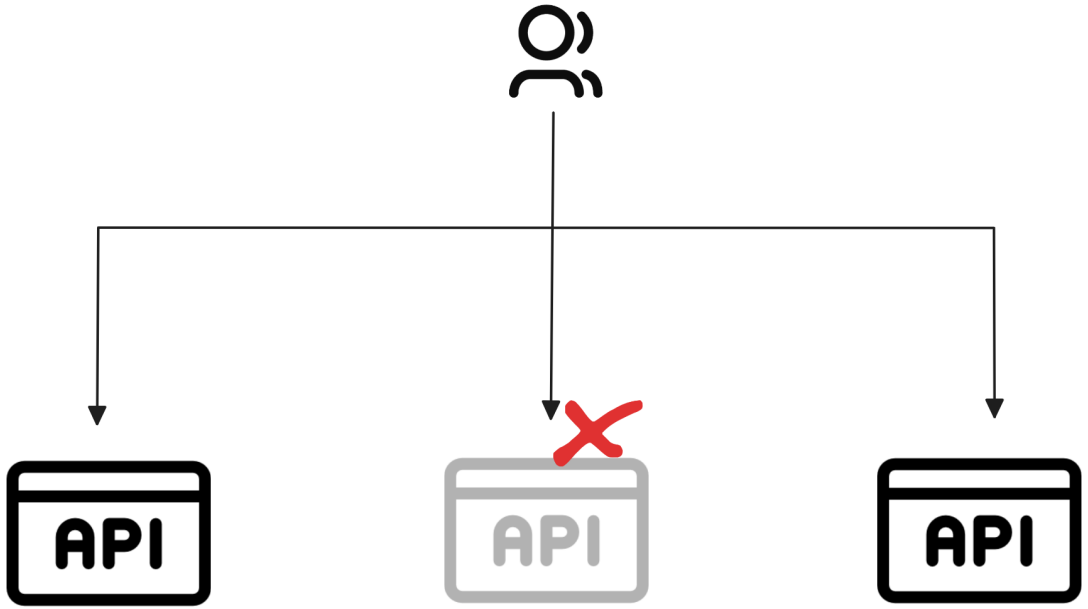
Docker multi server



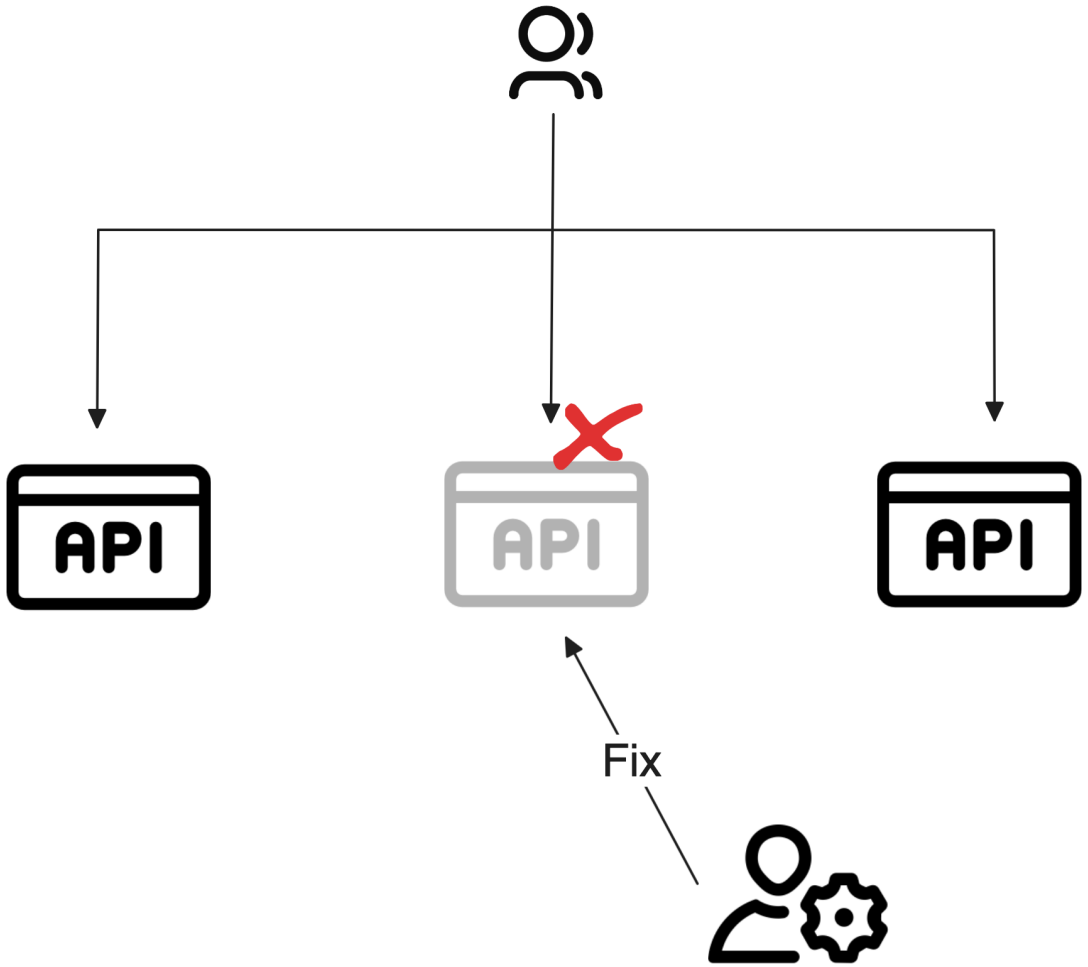
Applications problem



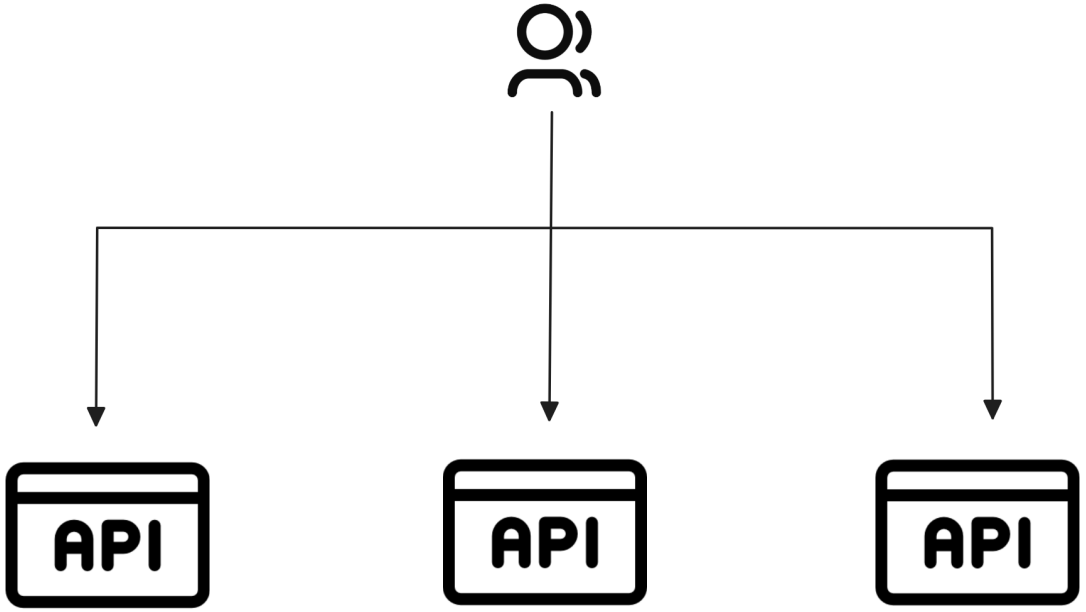
Applications problem



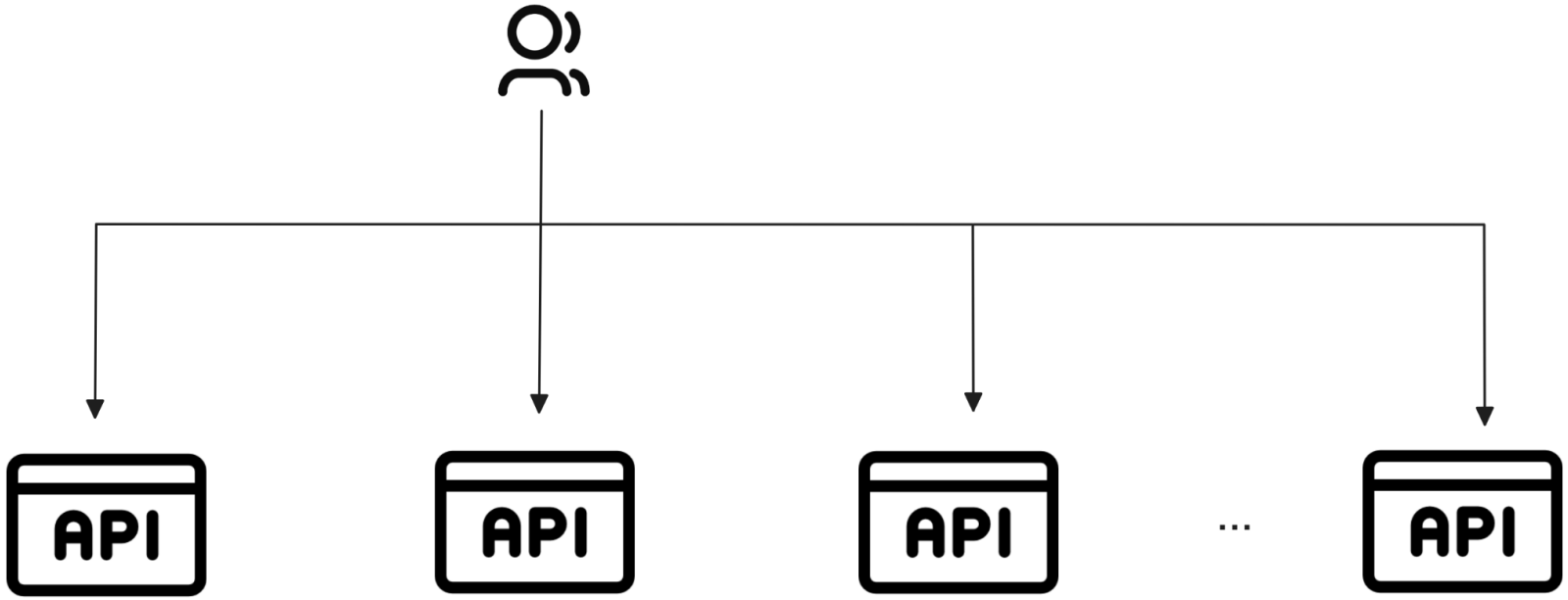
Applications problem



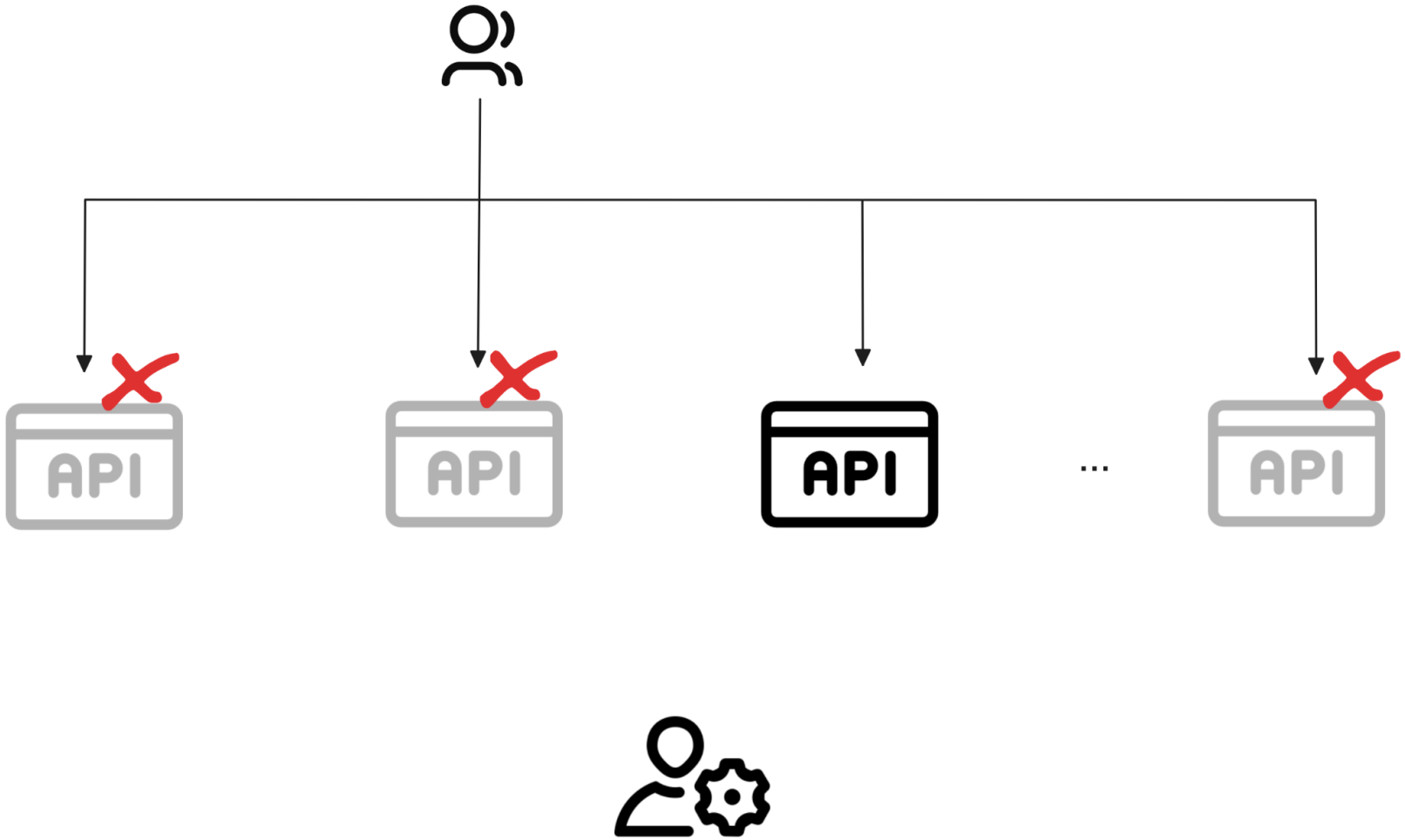
Applications problem



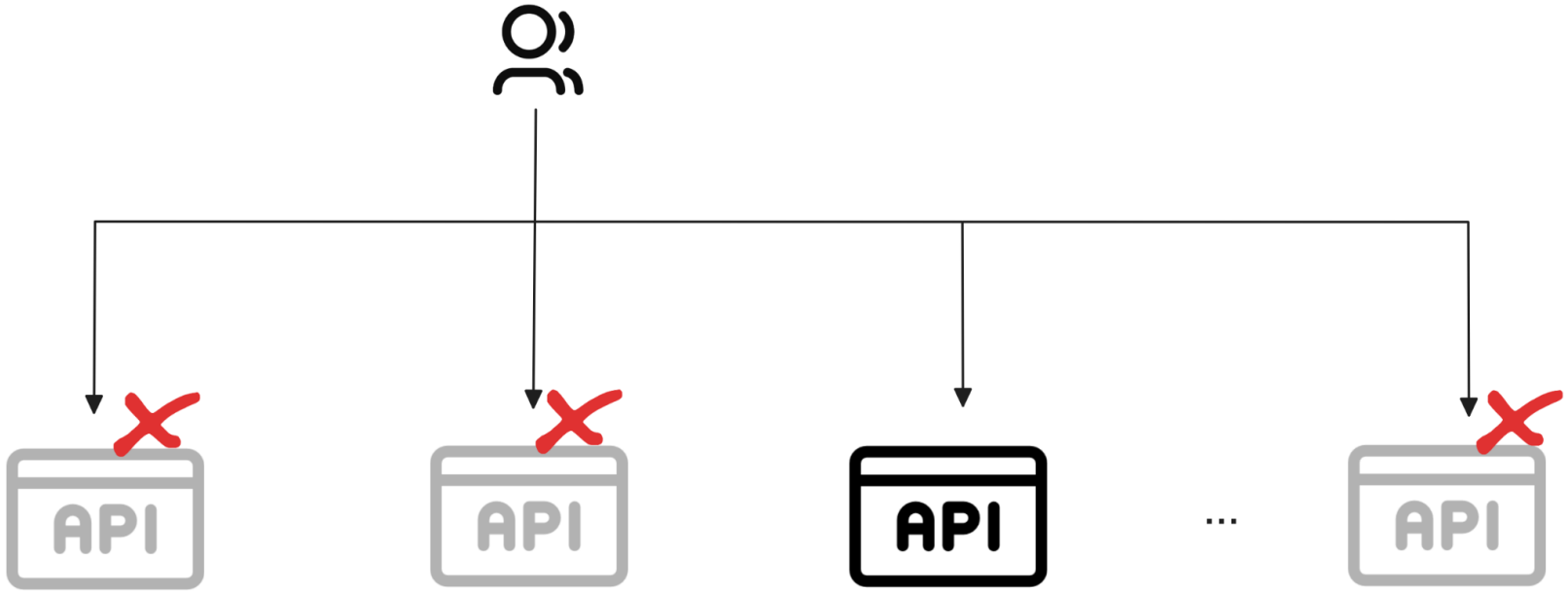
Applications problem



Applications problem



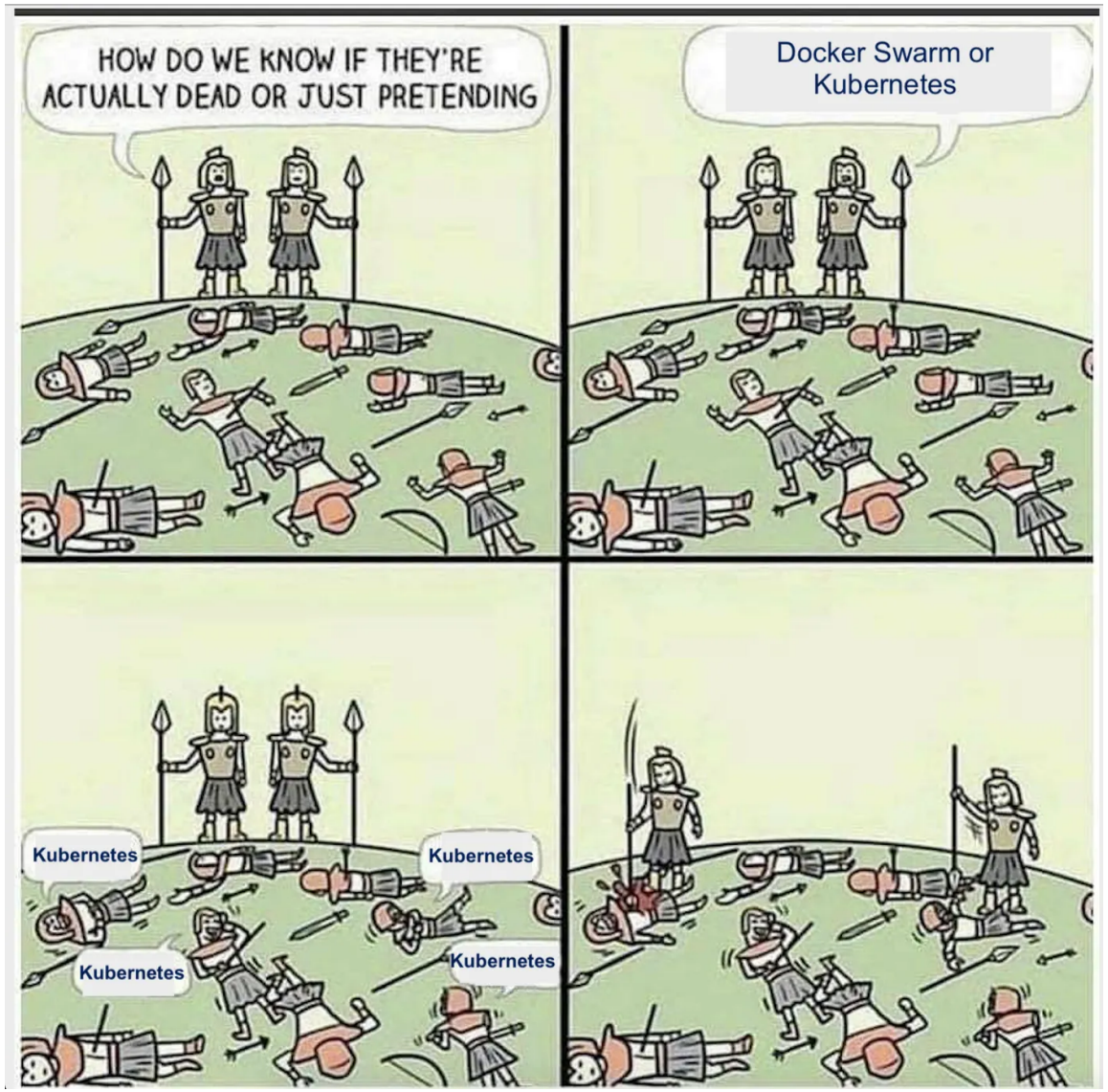
Applications problem



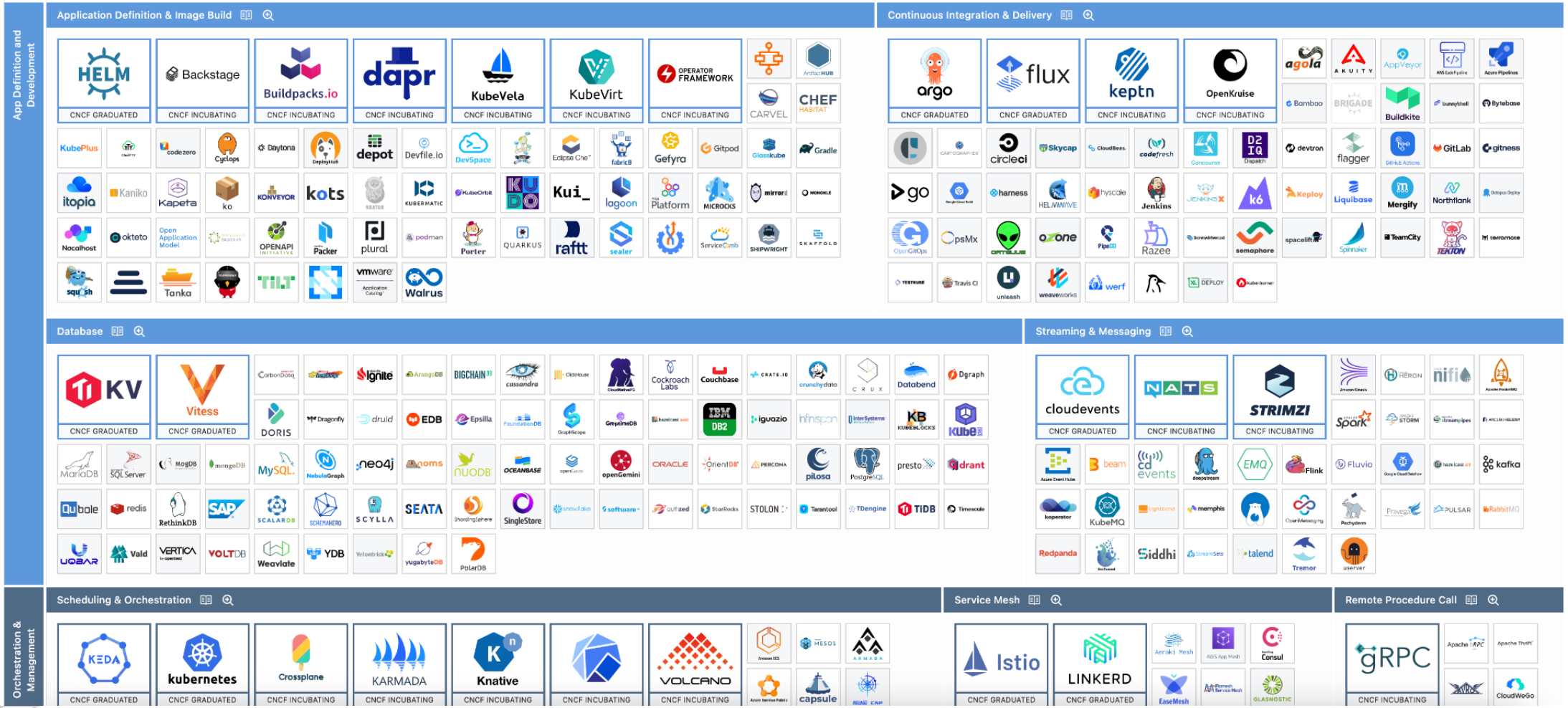
Fix !!!!

What is Kubernetes ?

- Kubernetes in Greek means the Helmsman
- Also known as k8s
- Inspired and informed by Google’s experiences and internal systems
- 100% opensource, written in GO
- Container orchestrator, runs and manages containers
- Kubernetes v1.0 released on July 2015, join CNCF on March 2016
- Container choose Docker, Container Orchestrator choose Kubernetes



CNCF Landscape

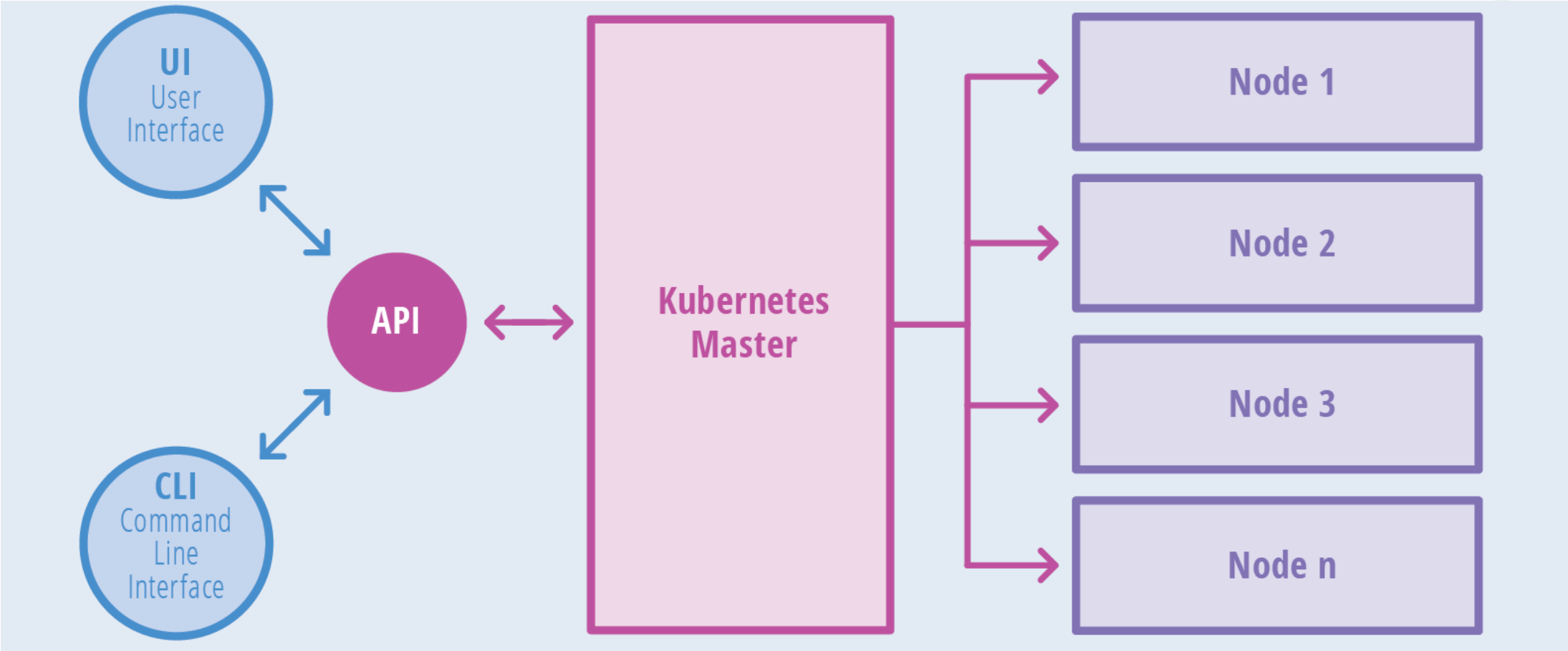


The features of Kubernetes

- Automatic bin packing
- Self healing
- Automated rollout & rollback
- Service Discovery & load Balancing
- Horizontal scaling
- Declarative Configuration

Kubernetes Architecture

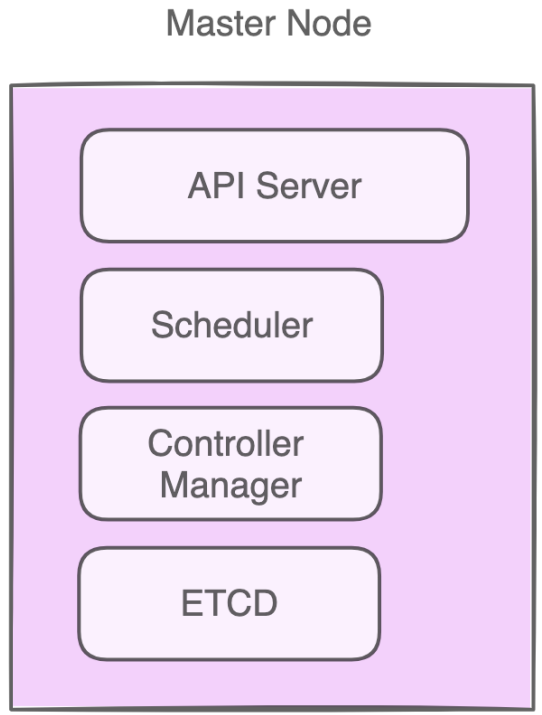
Kubernetes Architecture



Kubernetes Architecture

Master node

- Responsible for managing cluster
- Access master node via CLI, UI or API
- For fault tolerance, can be more than 1 master in cluster
- Include 4 components: API Server, Scheduler, Controller manager and ETCD



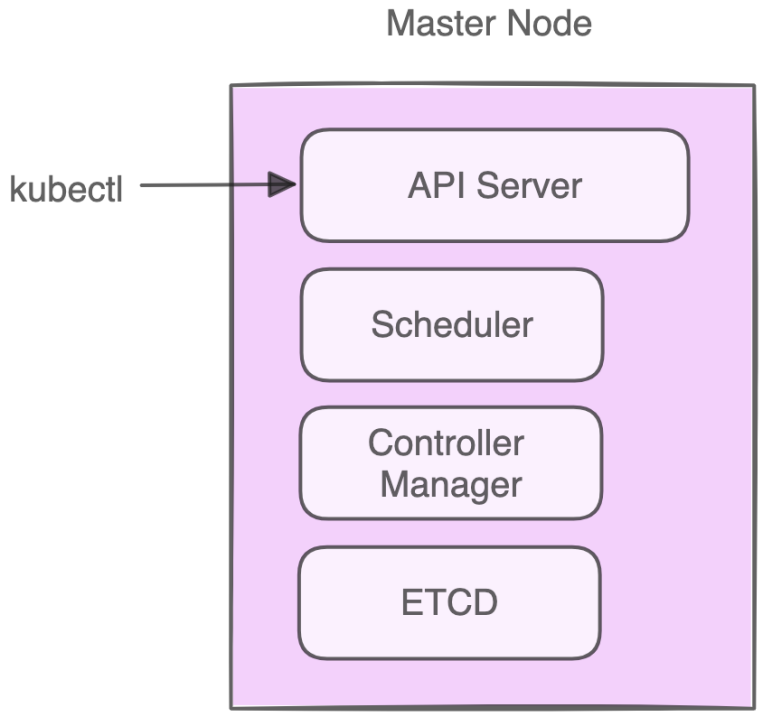
Kubernetes Architecture

Master node

API Server

- Centralized component where all the cluster components ar communicated

- Execute and validate from user command

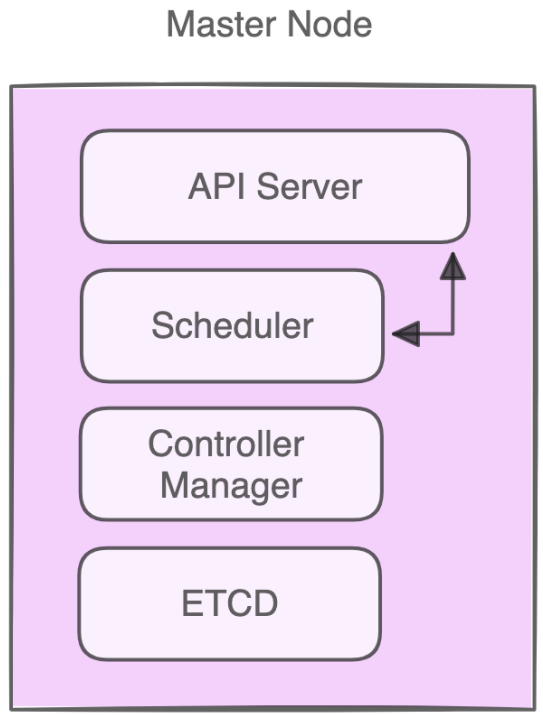


Kubernetes Architecture

Master node

Scheduler

- Assigning the application to worker node
- Automatically detect which pod to place on which node based on all factors

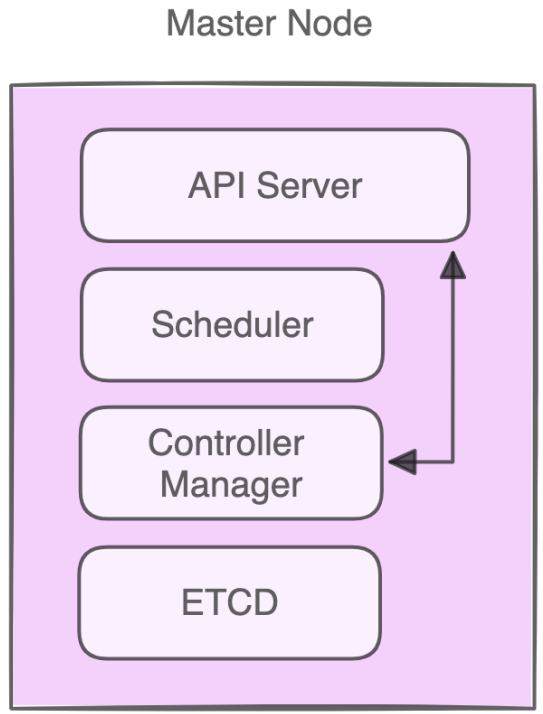


Kubernetes Architecture

Master node

Controller Manager

- The Controller Manager maintains the cluster
- Handles node failures, replicating components, maintaining the correct number of pods, etc.

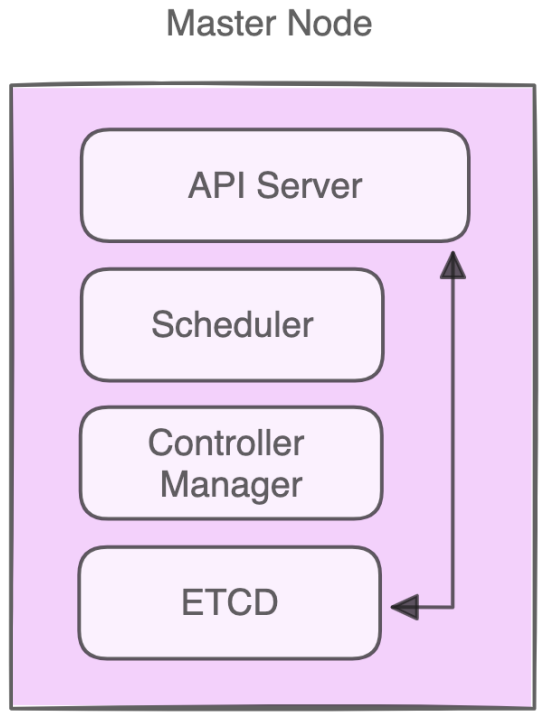


Kubernetes Architecture

Master node

ETCD

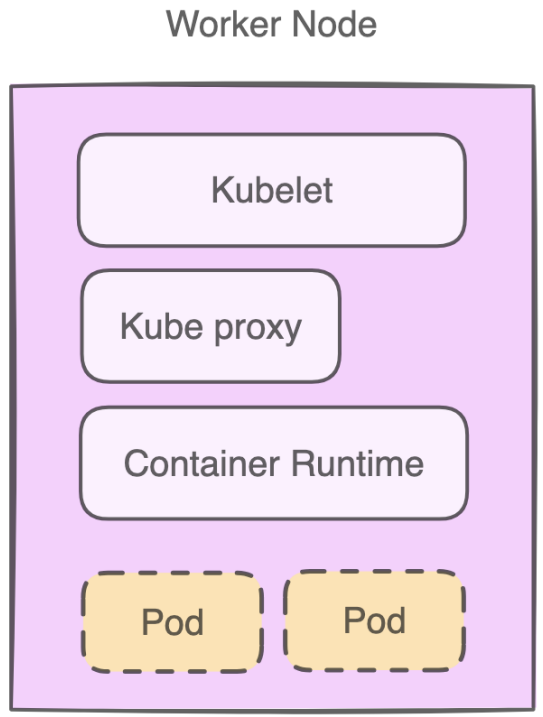
- ETCD is a distributed reliable key-value store use by Kubernetes to store all data used to manage the cluster
- When you have multi master node in cluster, ETCD stores all information in a distributed manner
- Application data doesn't store in ETCD



Kubernetes Architecture

Worker node

- Responsible about applications
- Include 3 components: **Container Runtime, Kubelet** and **Kube Proxy**

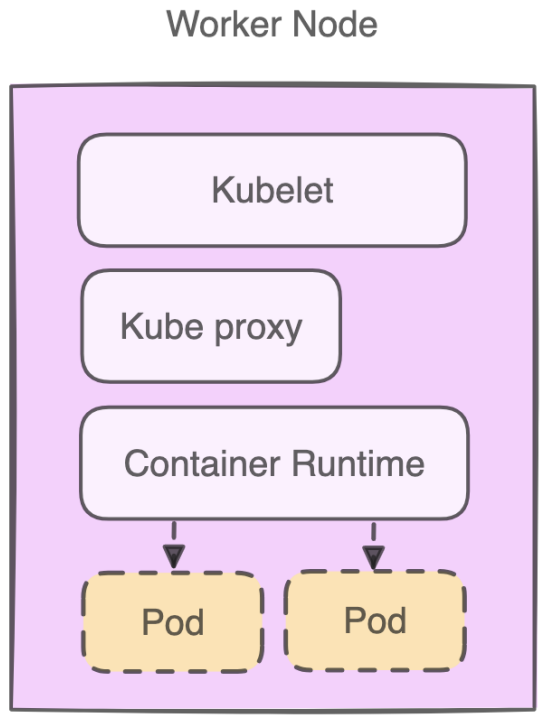


Kubernetes Architecture

Worker node

Container Runtime

- Container runtime which runs the containers like Docker, crio or containers
- Pulling the images and and run the containers

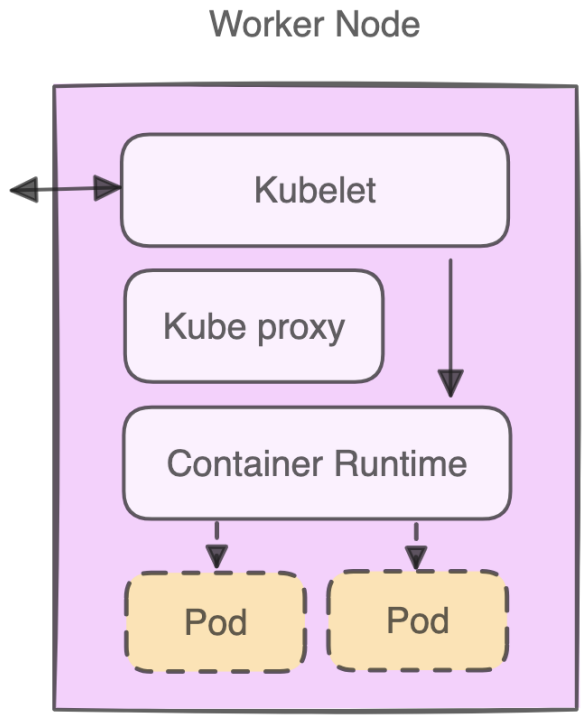


Kubernetes Architecture

Worker node

Kubelet

- Interacting with the master node
- Contact the container runtime for create pod depend on Po
- Provide health information of the worker node

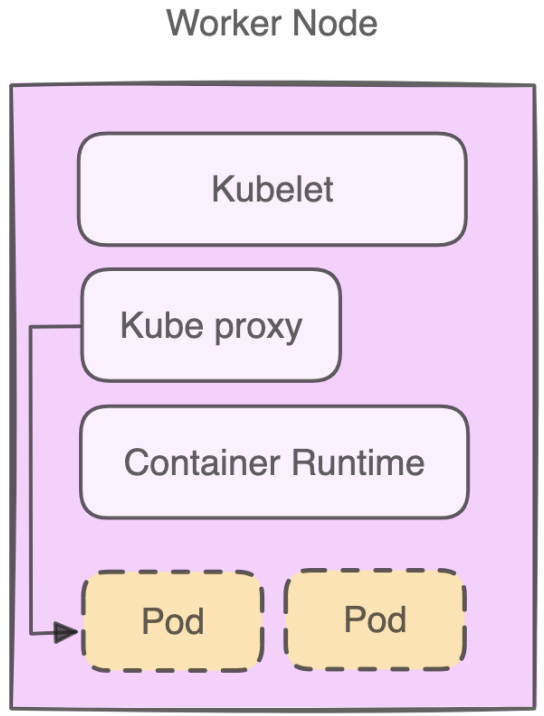


Kubernetes Architecture

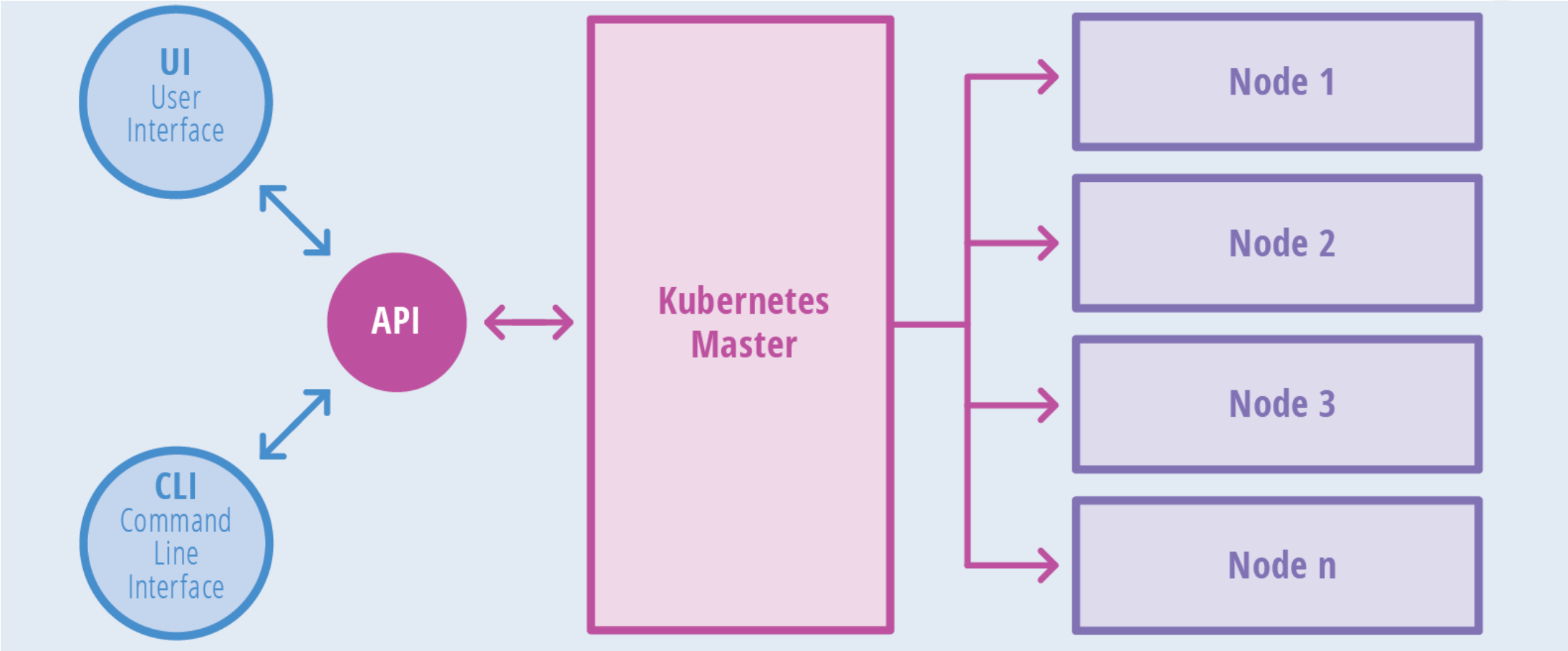
Worker node

Kube Proxy

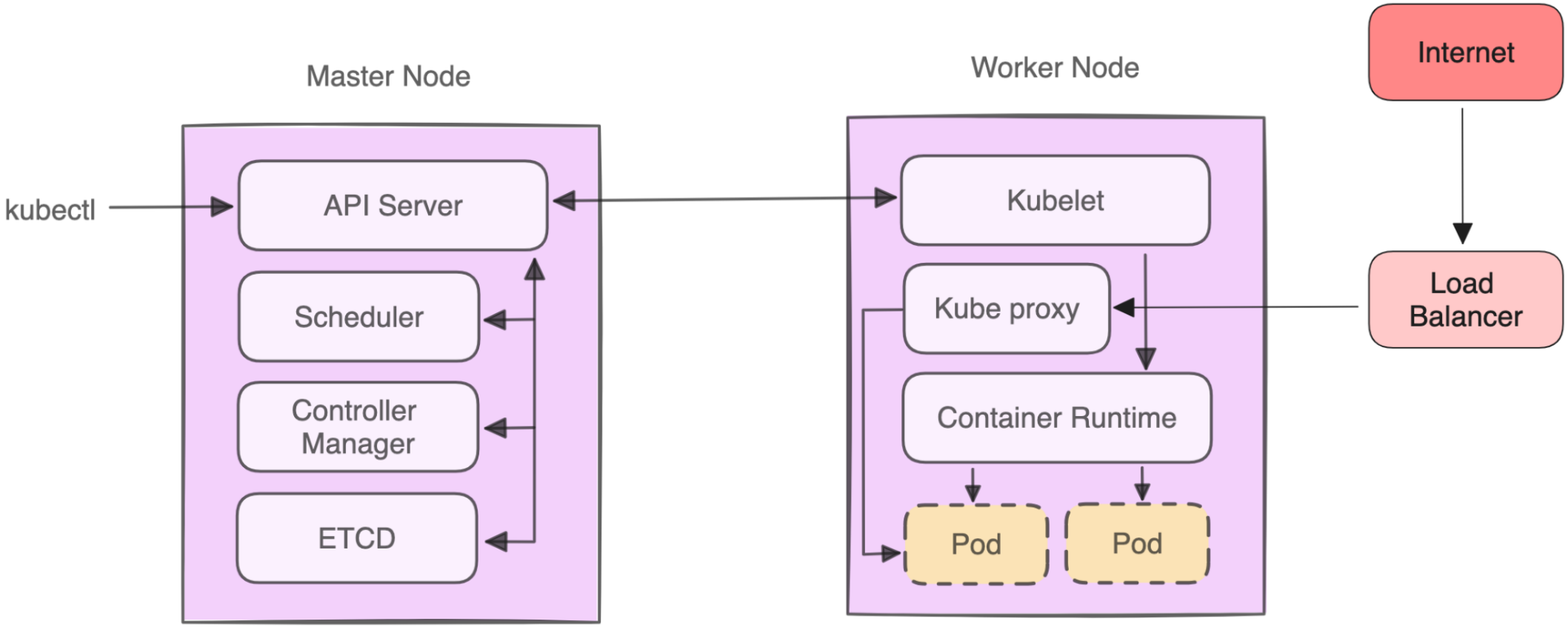
- Responsible for ensuring network traffic is routed properly to internal and external services
- Manage route for reduce latency time for example pod will communicate with another pod on same node first



Kubernetes Architecture



Kubernetes Architecture



Kubectl

Kubectl

- Command line
- Control the Kubernetes cluster manager
- CRUD Kubernetes resources

Kubectl example

- $ kubectl apply
- $ kubectl create
- $ kubectl get
- $ kubectl describe
- $ kubectl delete

Cluster

Cluster

- On Cloud
- On Premise
- On local development

Cluster on cloud

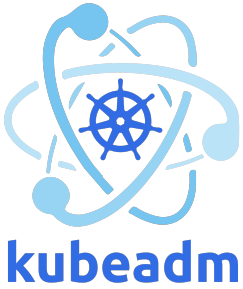
Amazon **EKS**

Google Kubernetes Engine

Azure Kubernetes Service (AKS)

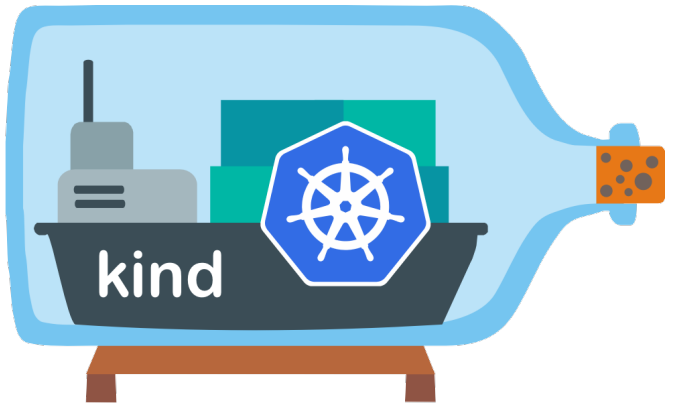
Alibaba Cloud

Cluster on premise



Cluster on local development

K3D



minikube

Cluster comparing

Feature	Kubernetes on Cloud	Kubernetes On-Premise
Setup & Maintenance	- Easy to set up with automated updates and patches - Minimal operational overhead	- Requires manual setup - Ongoing maintenance and upgrades are the user's responsibility
Cost	- Hight depend cloud provider - Costs tied to usage - No upfront infrastructure investment	- Lower costs - High initial investment for hardware and infrastructure
Scalability	- Highly scalable with near-infinite resources. - Automatic scaling options available	- Limited by on-premise hardware resources. - Scaling may require additional hardware and setup.
Disaster Recovery	- Built-in disaster recovery and backups.	- Must design and implement your own
Customization	- Limited by cloud provider's offerings and configurations.	- Full control over configurations and infrastructure.

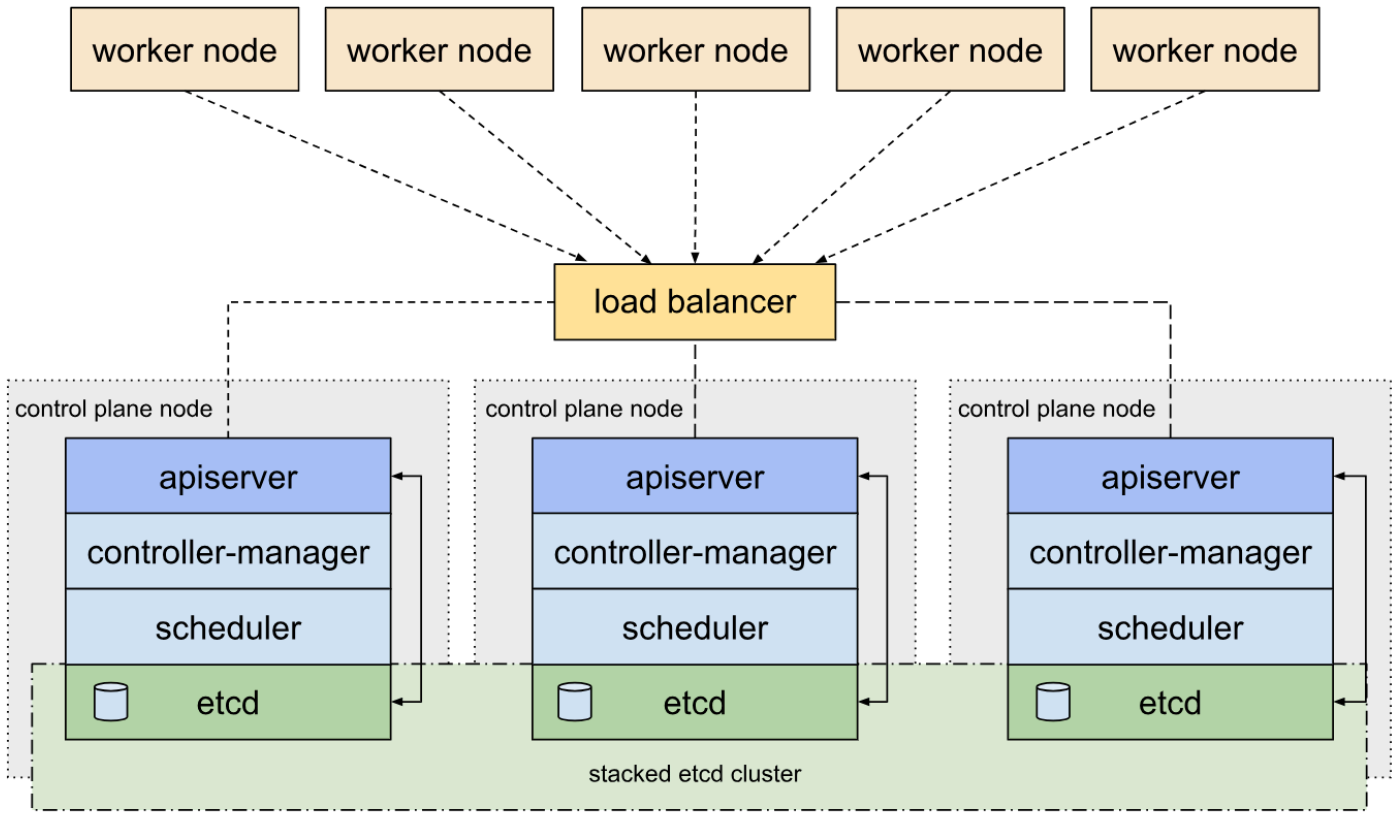
Cluster comparing

Feature	Kubernetes on Cloud	Kubernetes On-Premise
Security	- Security managed by cloud providers	- Full control over security configurations
Networking	- Advanced networking features available (e.g., VPCs, load balancers). - Simplified setup with integration to other cloud services	- Full control over network configurations. - Requires in-house expertise for complex networking setups
Upgrades & New Features	- Automatic access to the latest features and updates	- Full control over when and how to upgrade - Delayed to the latest features unless manually updated

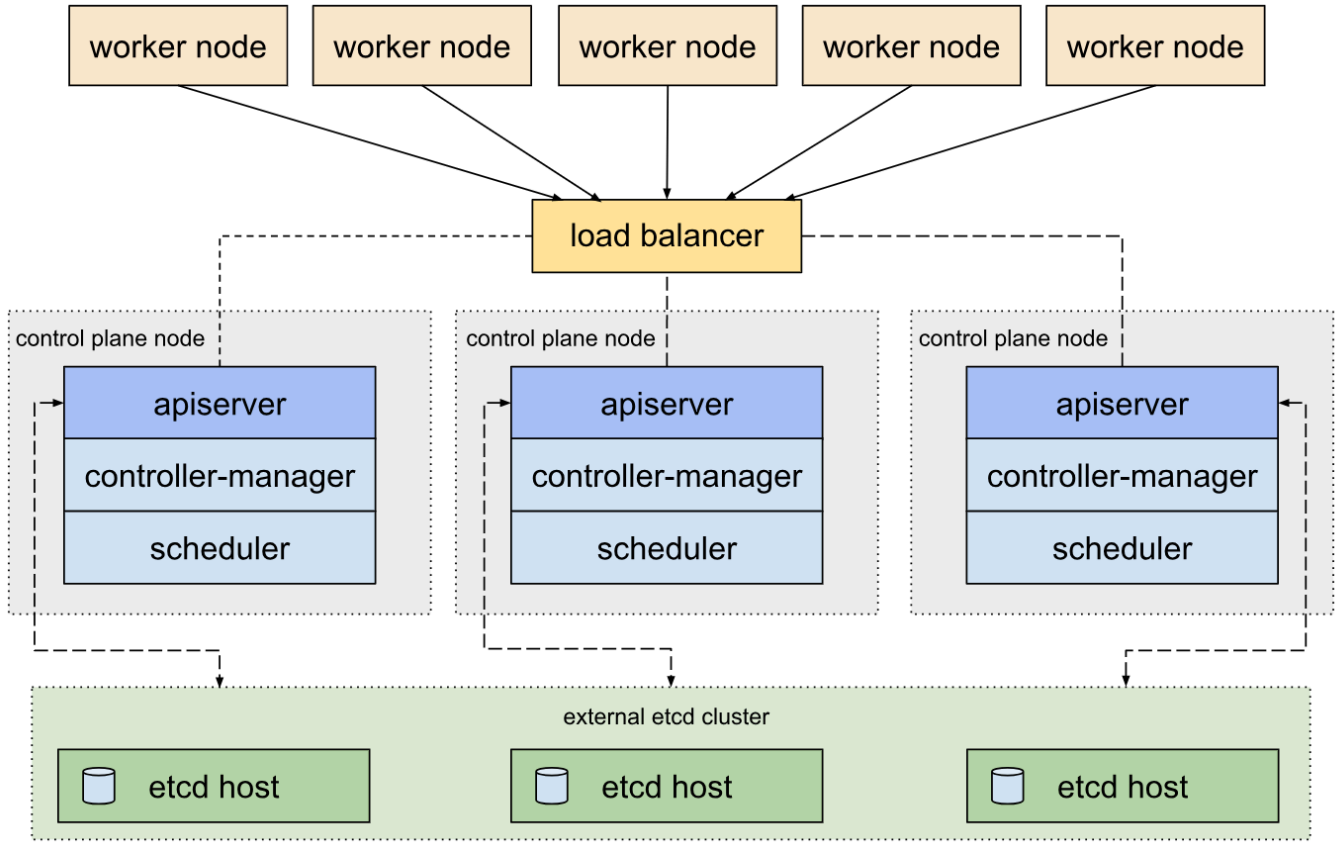
Options for Highly Available

- Stacked etcd topology
- External etcd topology

Stacked etcd topology



External etcd topology



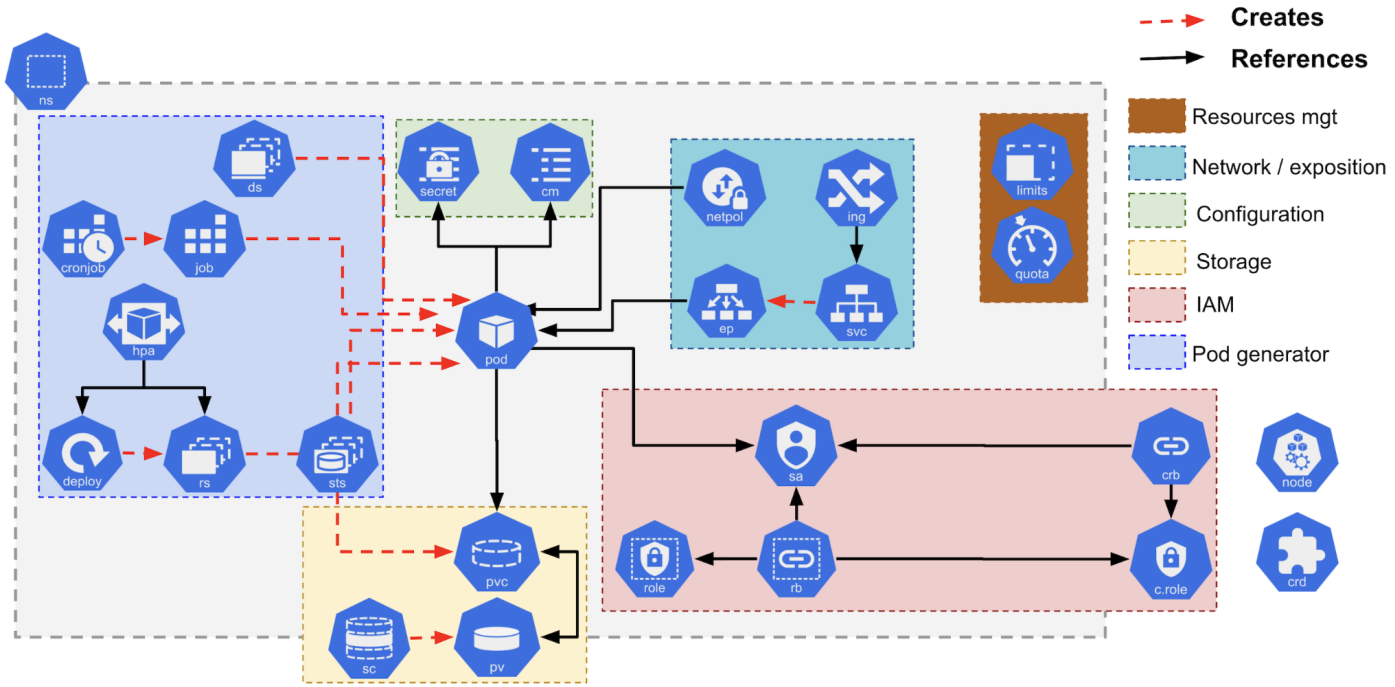
Fault Tolerance Table

Cluster Size	Majority	Failure Tolerance
1	1	0
2	2	0
3	2	**1**
4	3	1
5	3	**2**
6	4	2
7	4	**3**
8	5	3
9	5	**4**

Resources / Objects

Resources / Objects

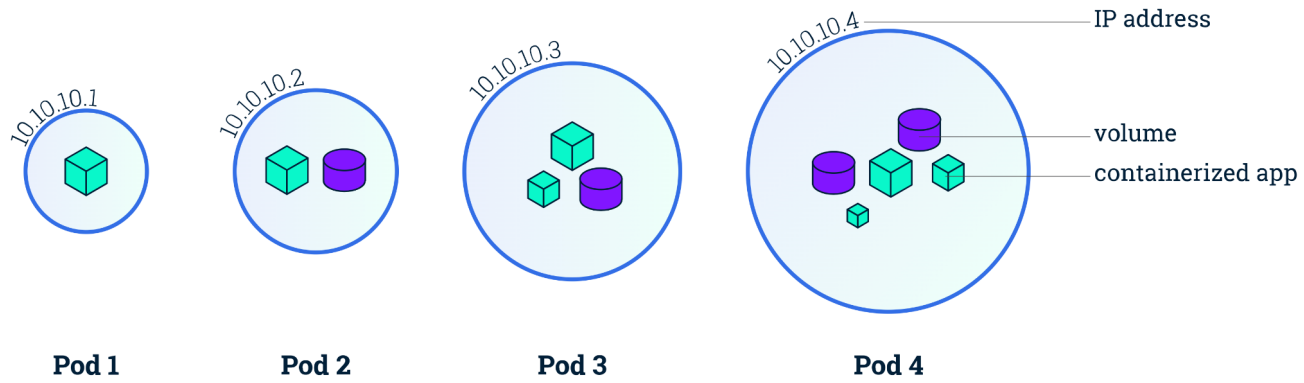
- Pod
- Deployment
- Service
- Ingress
- Config map
- Secret
- PV & PVC



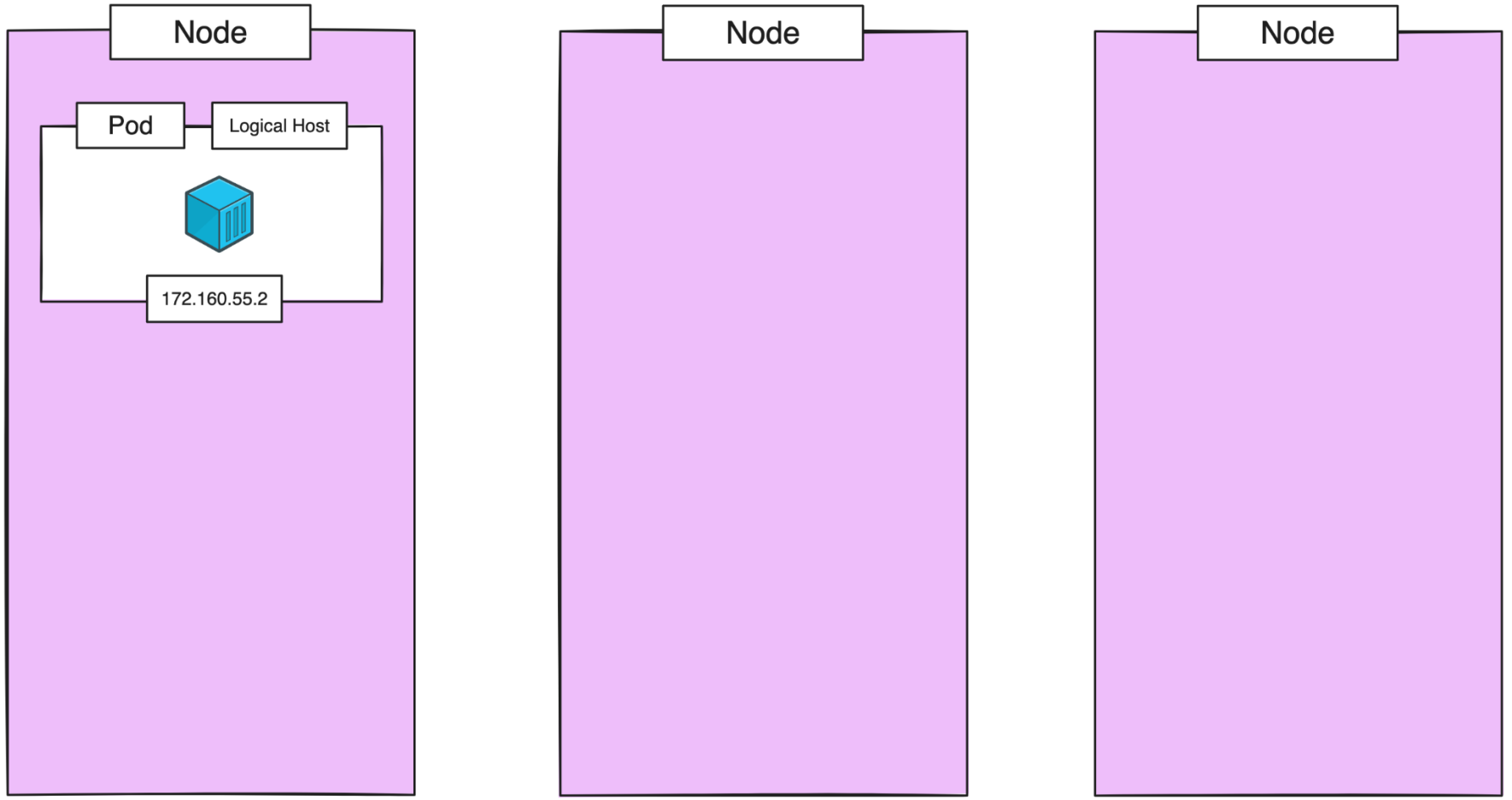
Pod

Pod

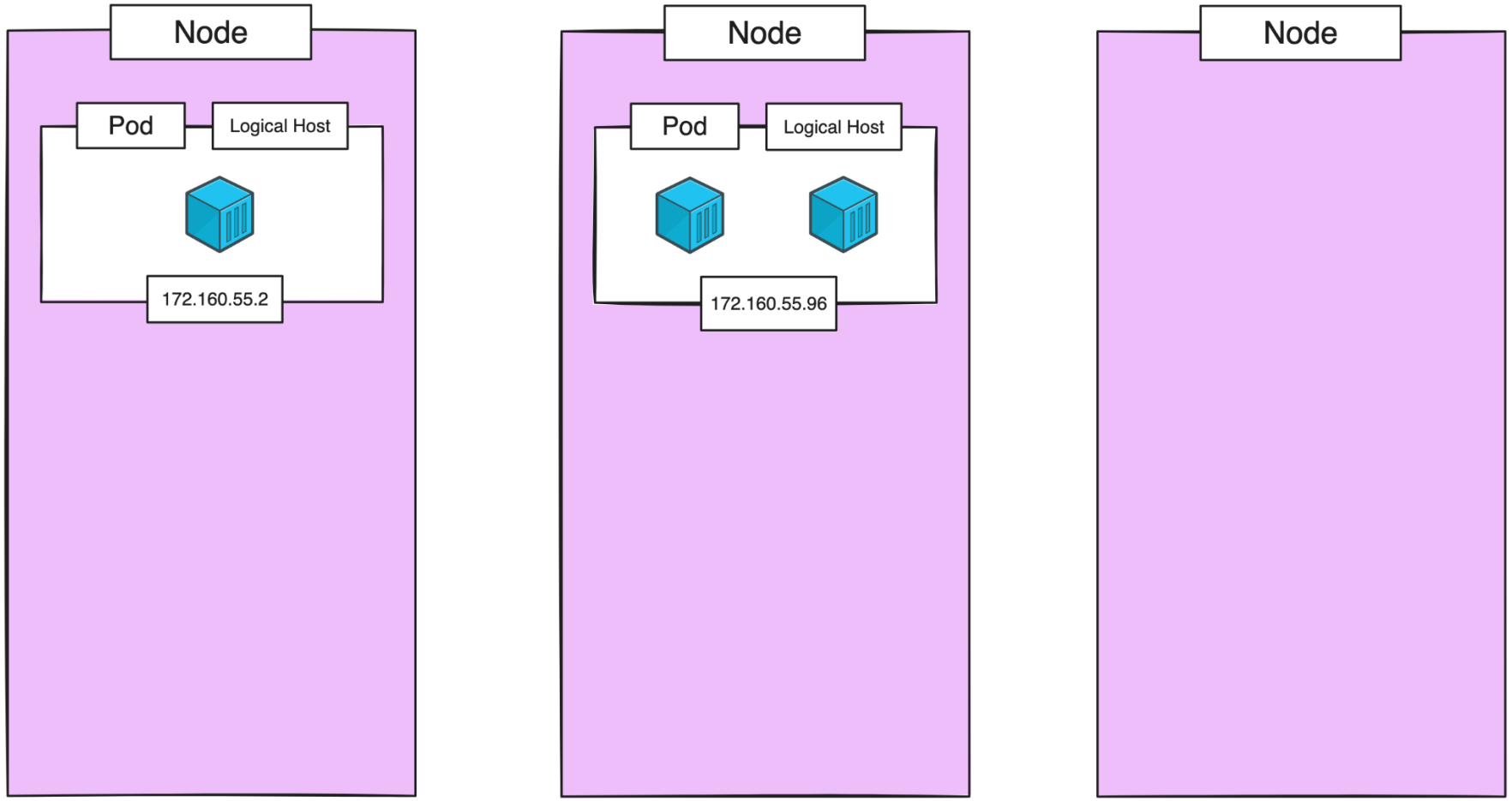
- Smallest deployable units of computing that you can create and manage in Kubernetes
- Logical Application
- One or more container in Pod (recommend 1 container per pod)
- Any containers in same pod will share storage volumes and network resources
- Each pod is assigned a unique ip



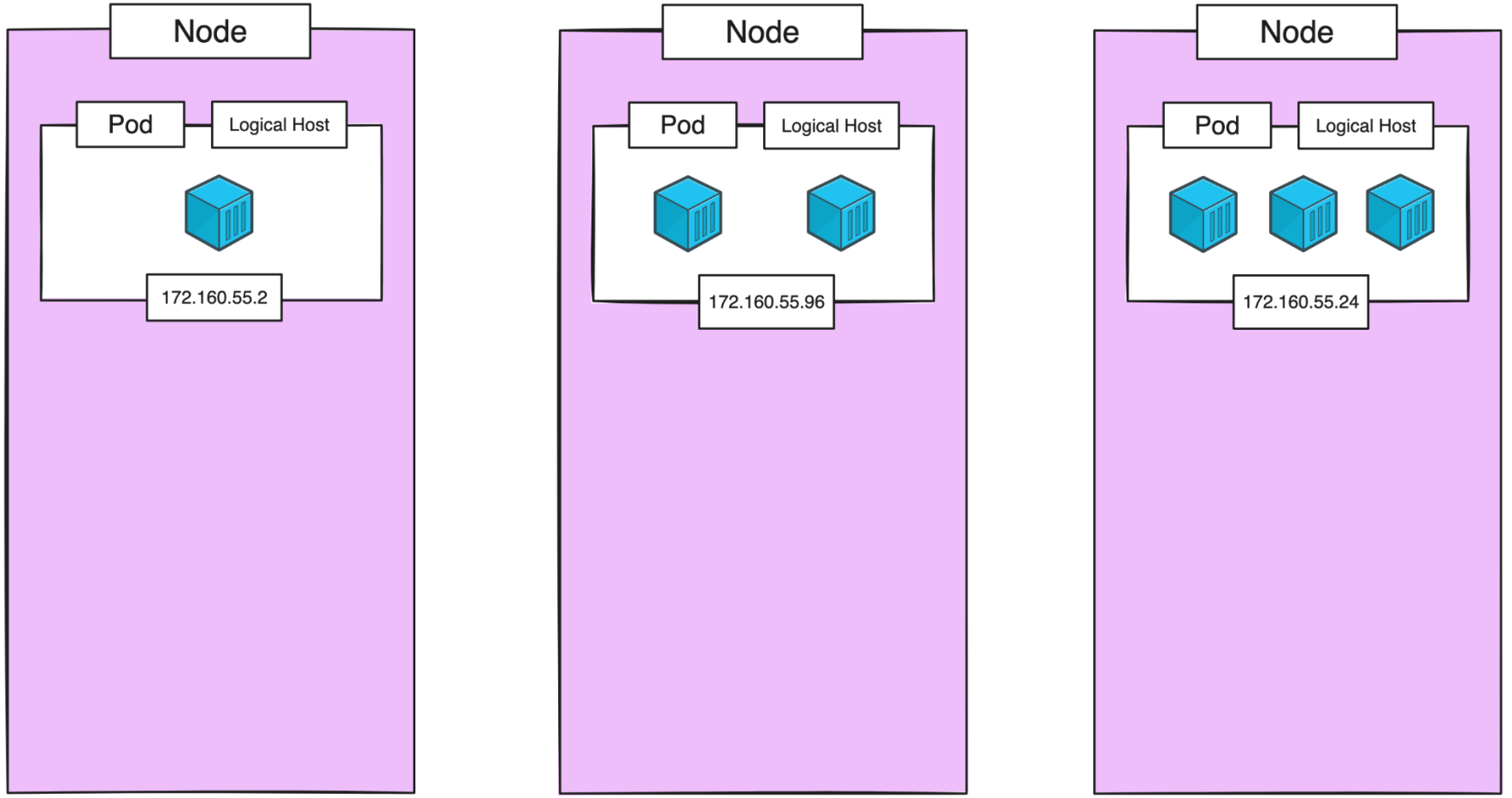
Node & Pod



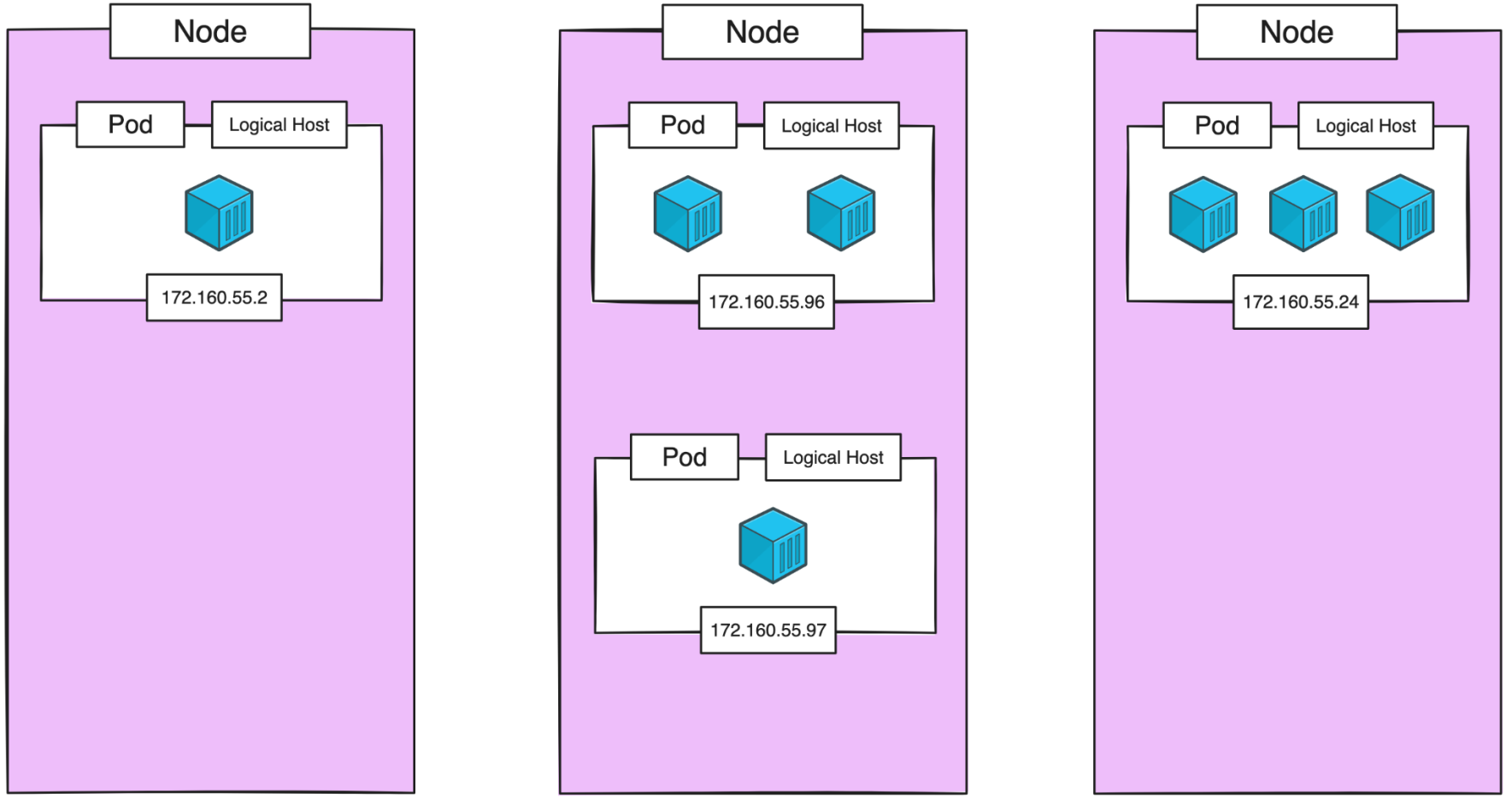
Node & Pod



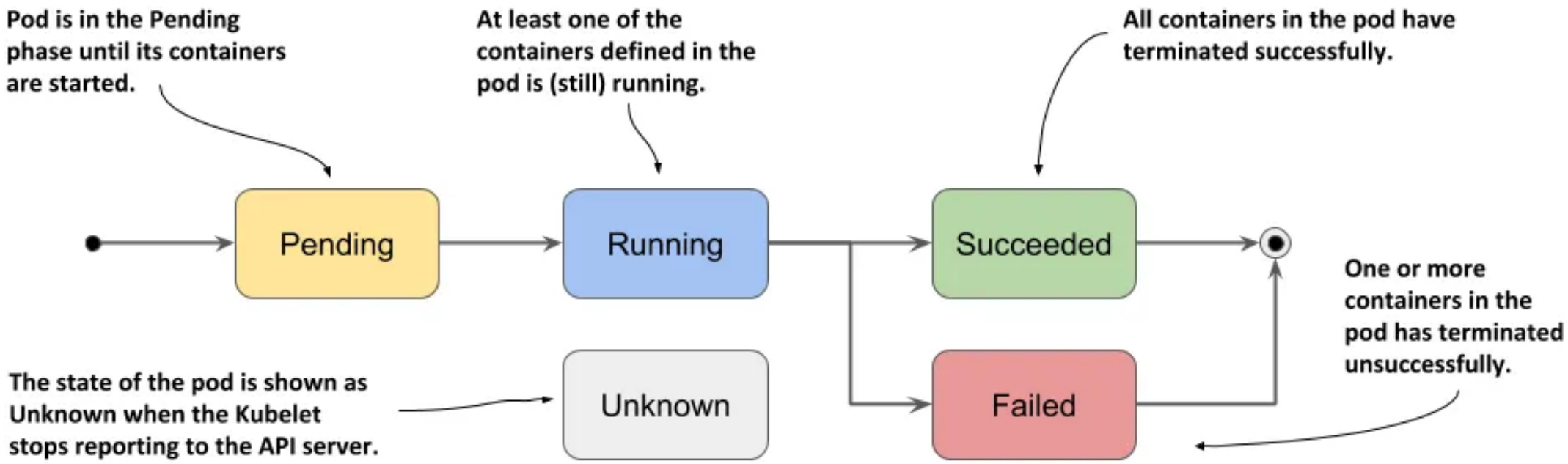
Node & Pod



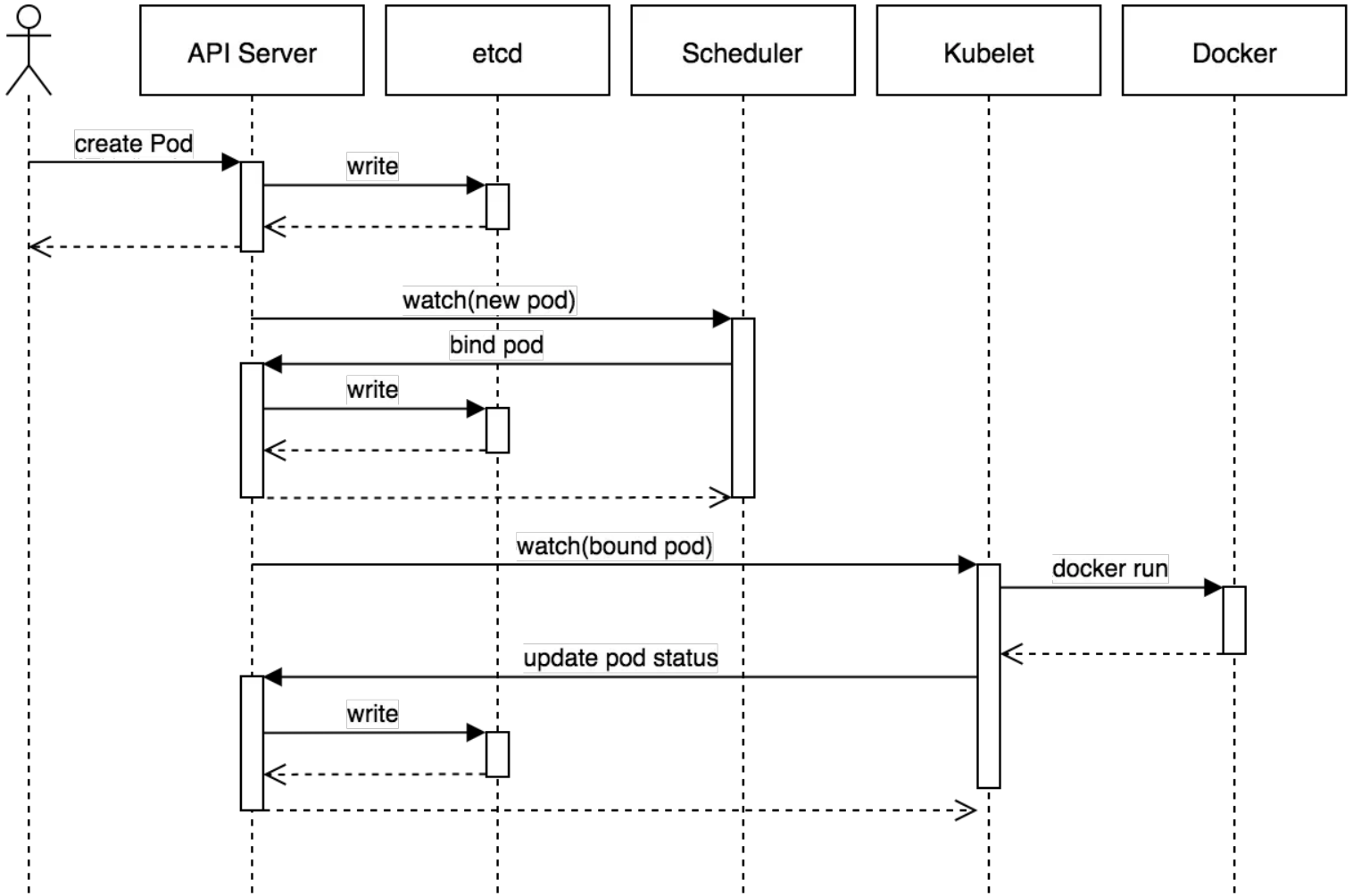
Node & Pod



State of Pod



Creation of Pod

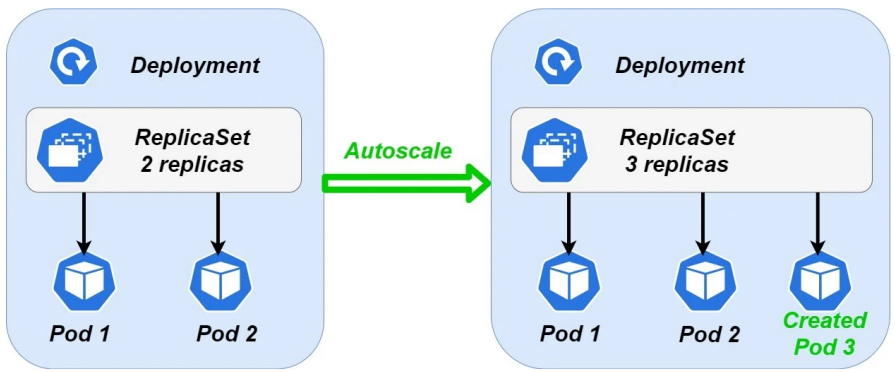


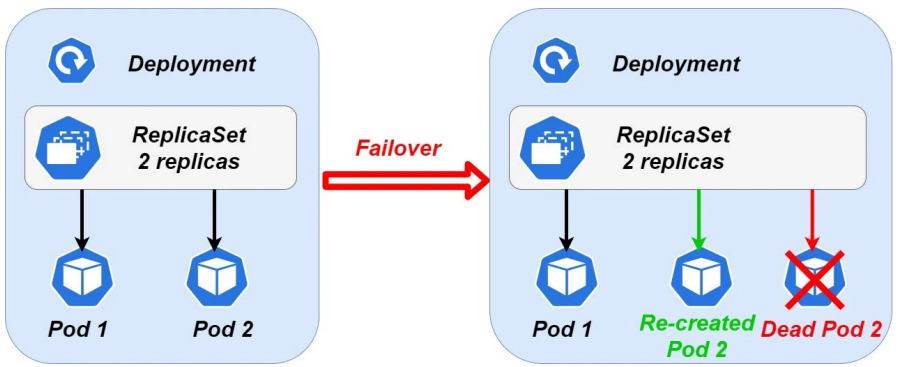
Workshop pod

Deployment

Deployment

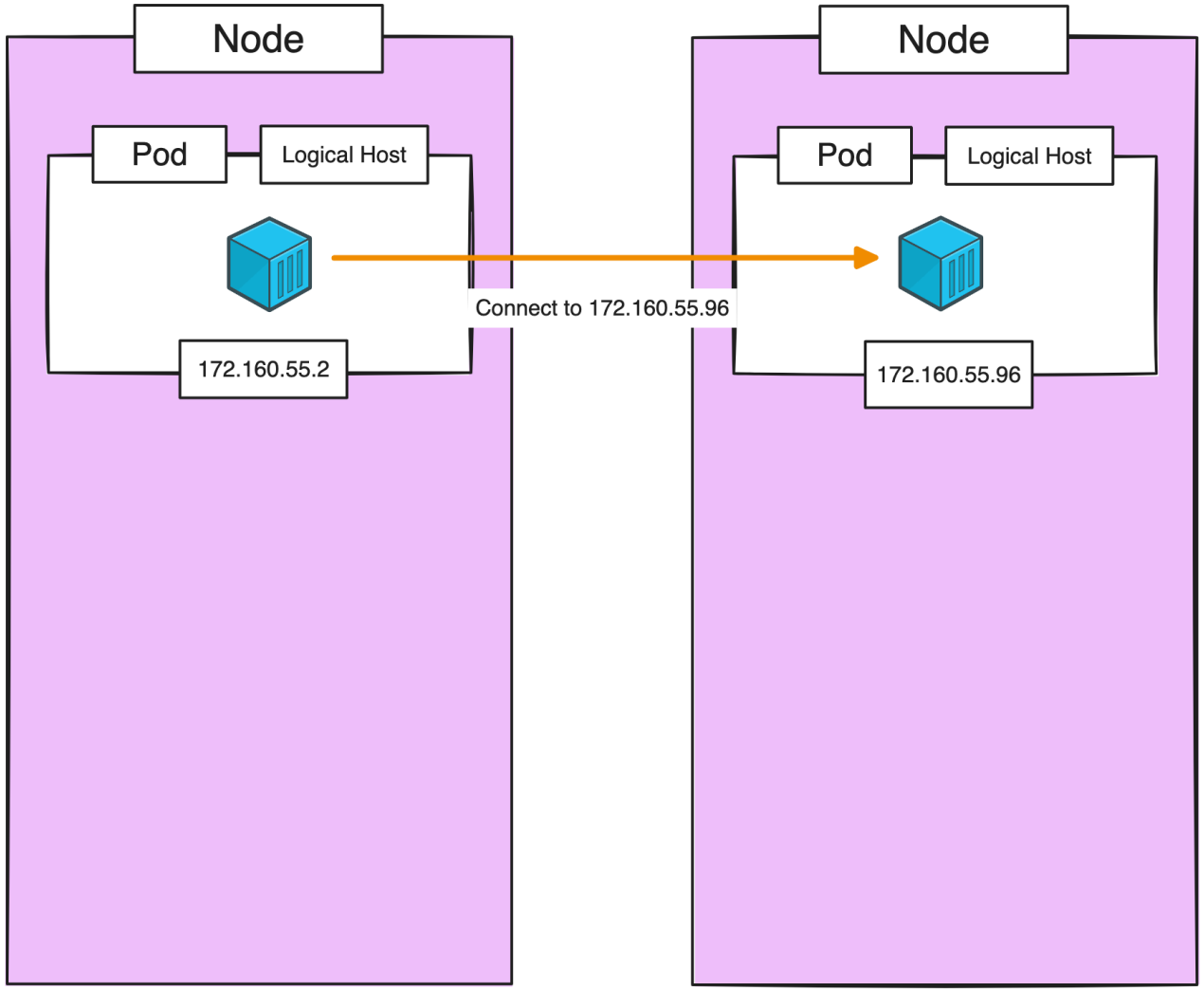
- A Deployment provides declarative updates for Pods and ReplicaSets
- You describe a **desired state** in a Deployment, and the Deployment Controller **changes the actual state to the desired state** at a controlled rate
- Manage for rollout and rollback
- Deployment => ReplicaSet => pod



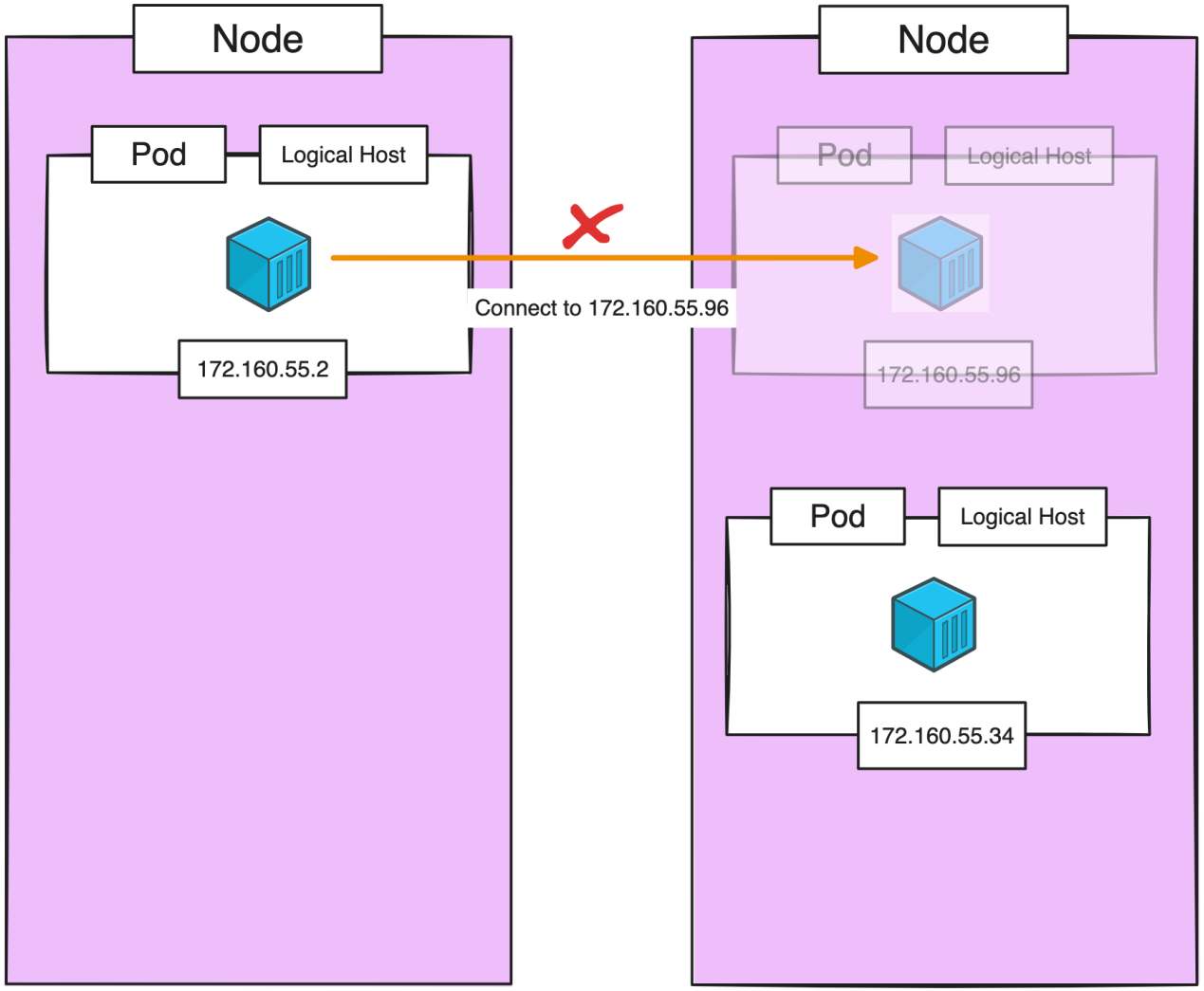


Workshop deployment

Problems



Problems

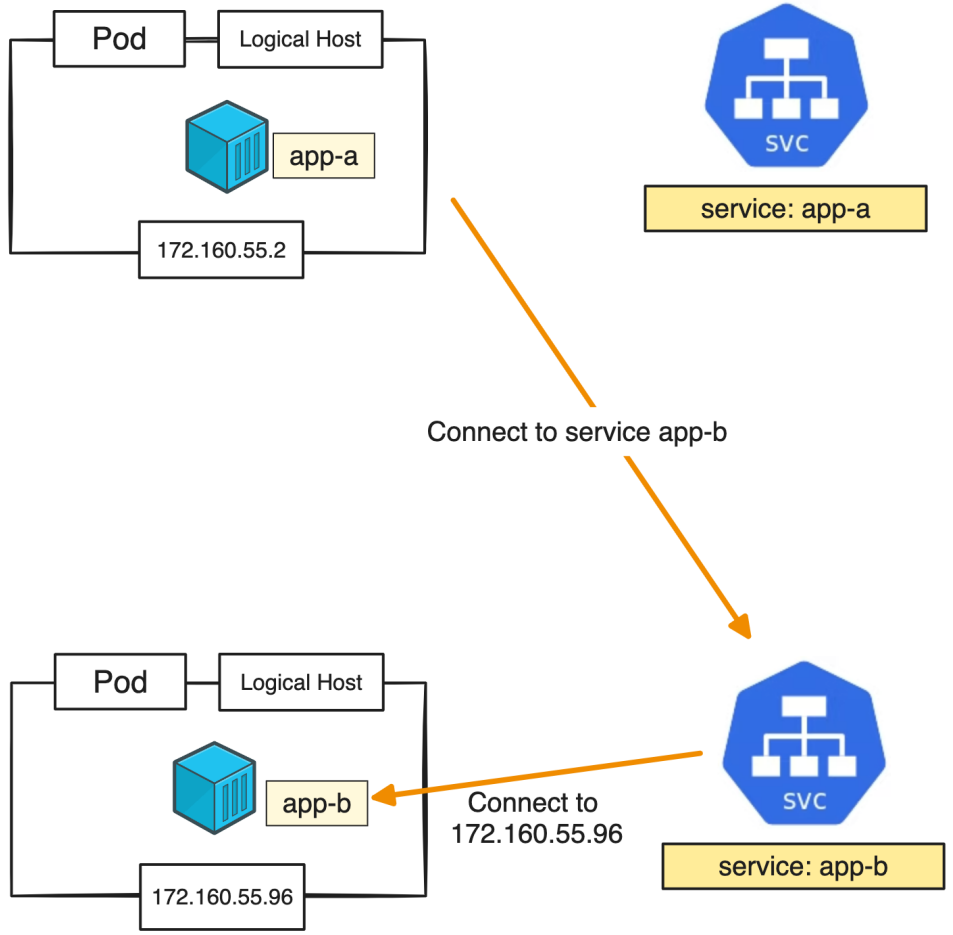


Service

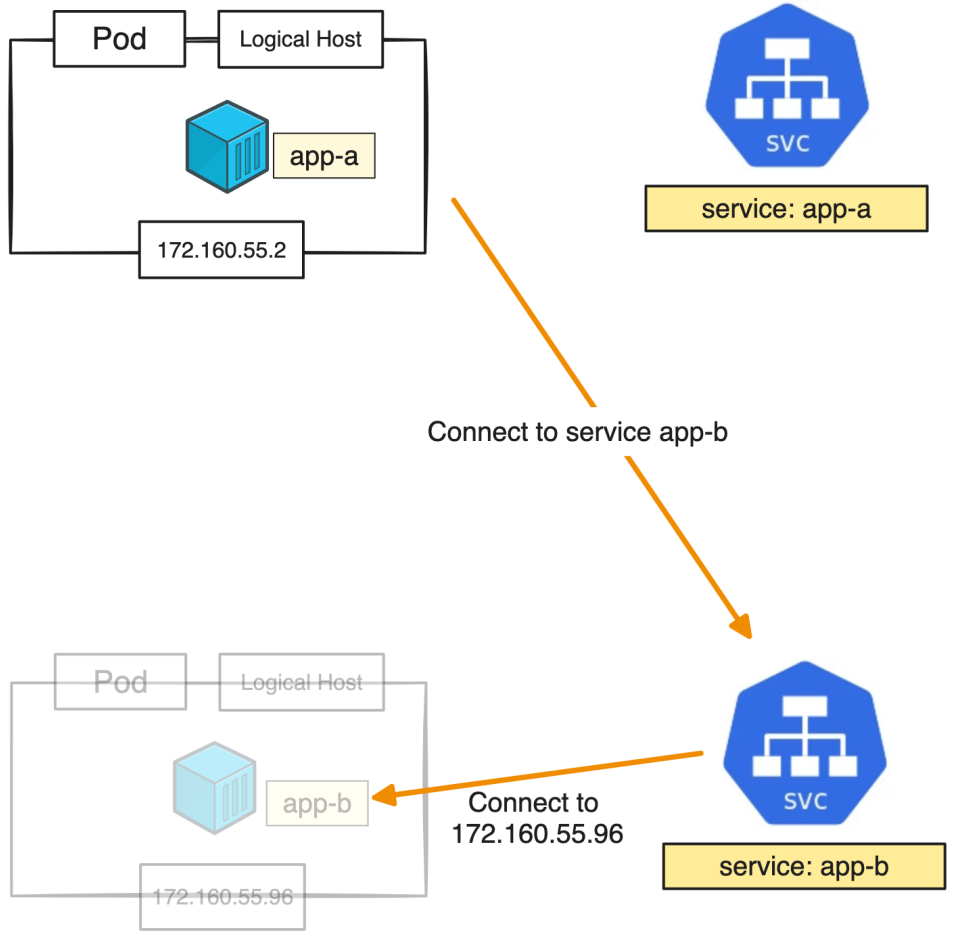
Service

- Acts as a virtual endpoint for your application
- Providing a stable and discoverable way to access your pods without worrying about their individual IP addresses or network details
- Logical load balancer
- Communicating for outside and inside of cluster
- 4 types:
  - ClusterIP
  - NodePort
  - LoadBalancer
  - ExternalName

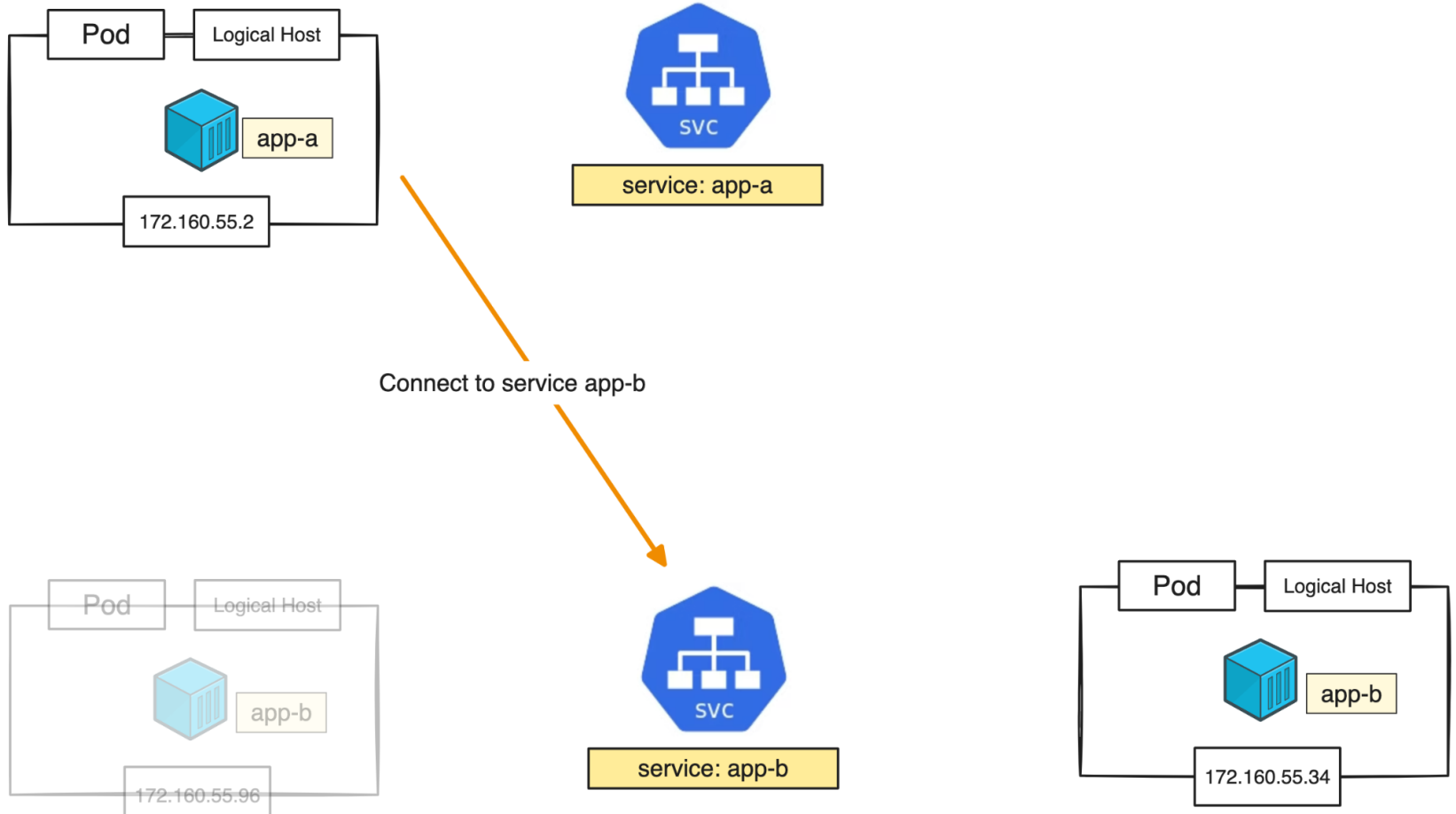
Service: Stable endpoint



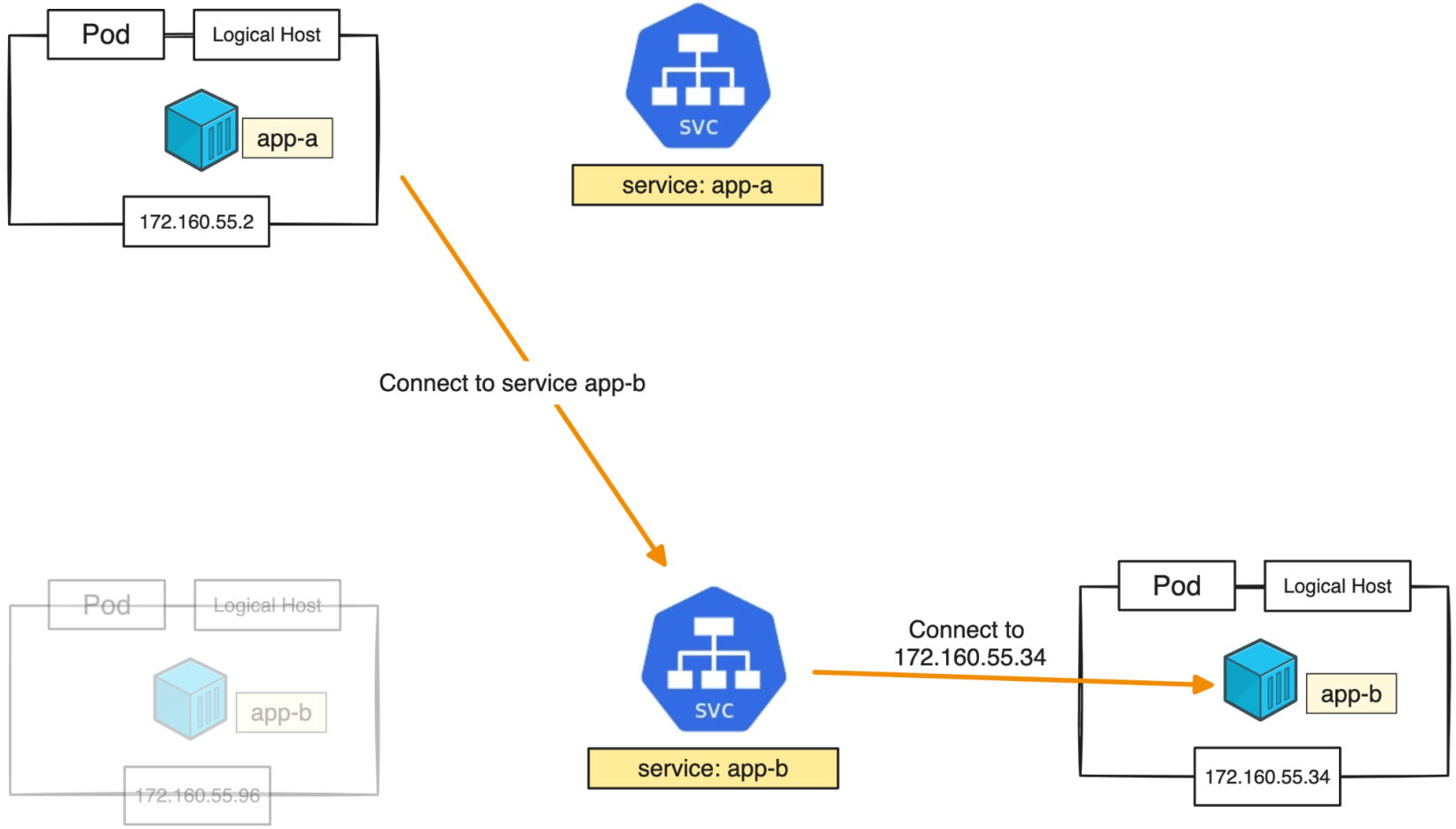
Service: Stable endpoint



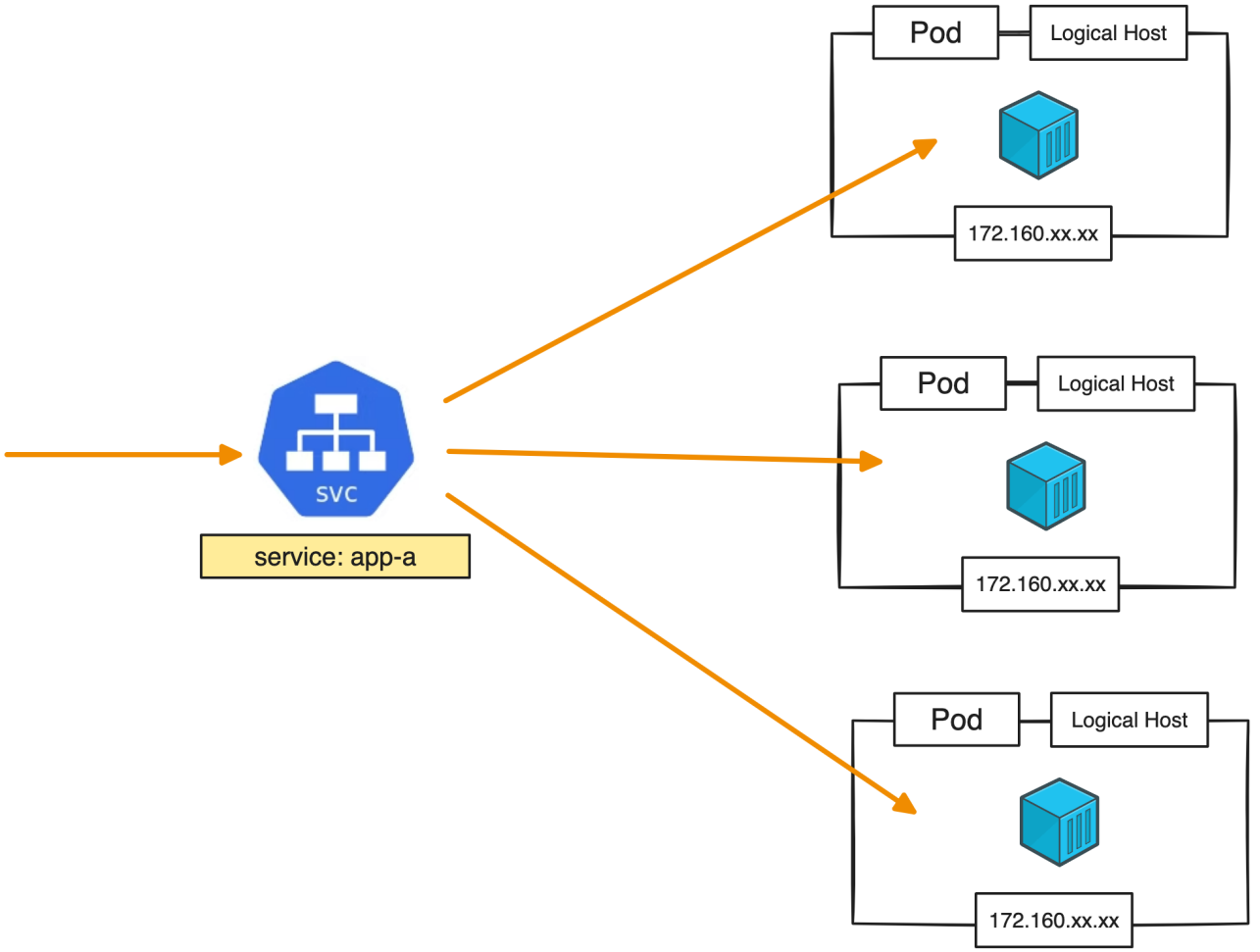
Service: Stable endpoint



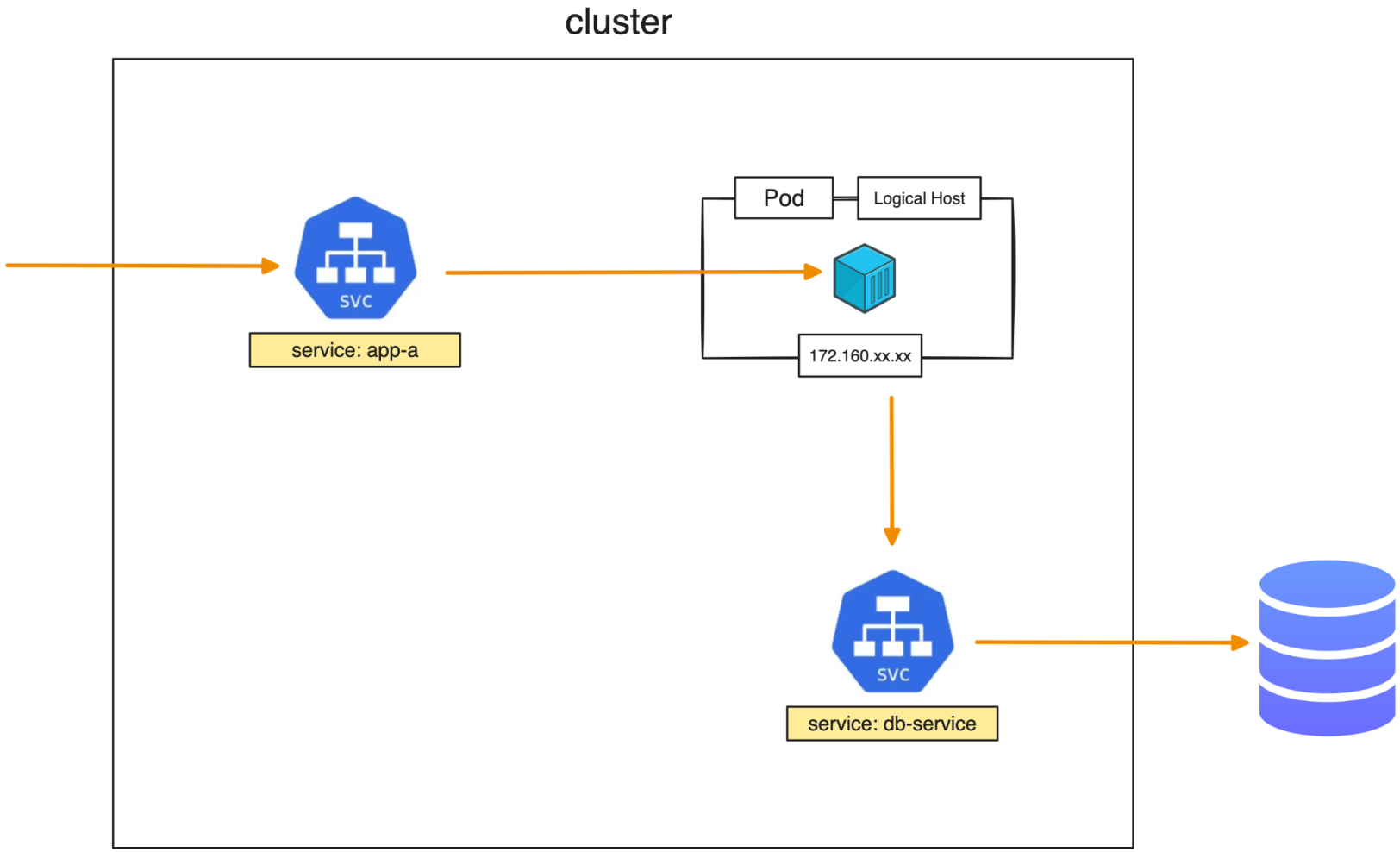
Service: Stable endpoint



Service: Load balancer

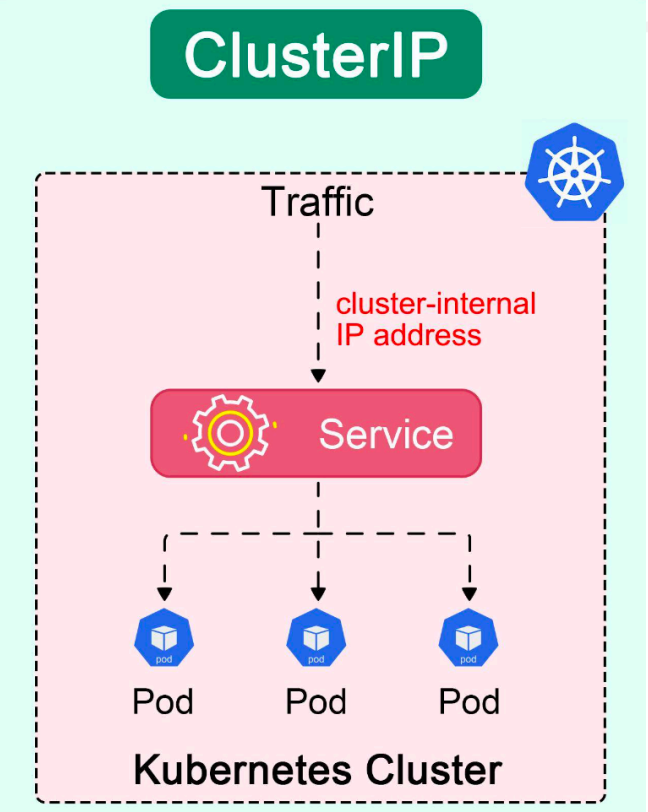


Service: Communicating inside & outside of cluster



Service type: ClusterIP

- ClusterIP is the default service type in Kubernetes
- It exposes the service on an internal IP address reachable **only within** the cluster



Advantages

- Suitable for inter-pod communication and internal services

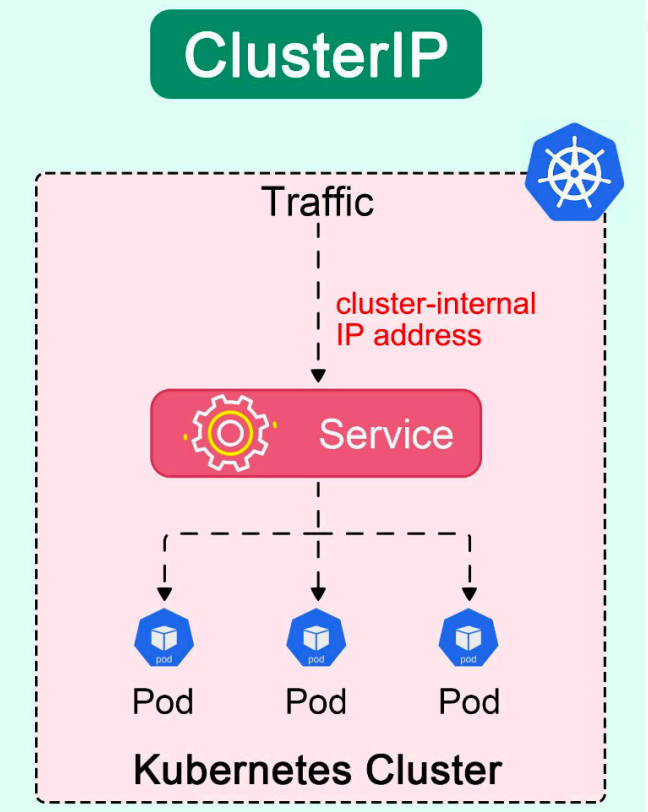
Disdvantages

- Limited to internal cluster communication
- Not accessible externally

Service type: ClusterIP

Use Cases

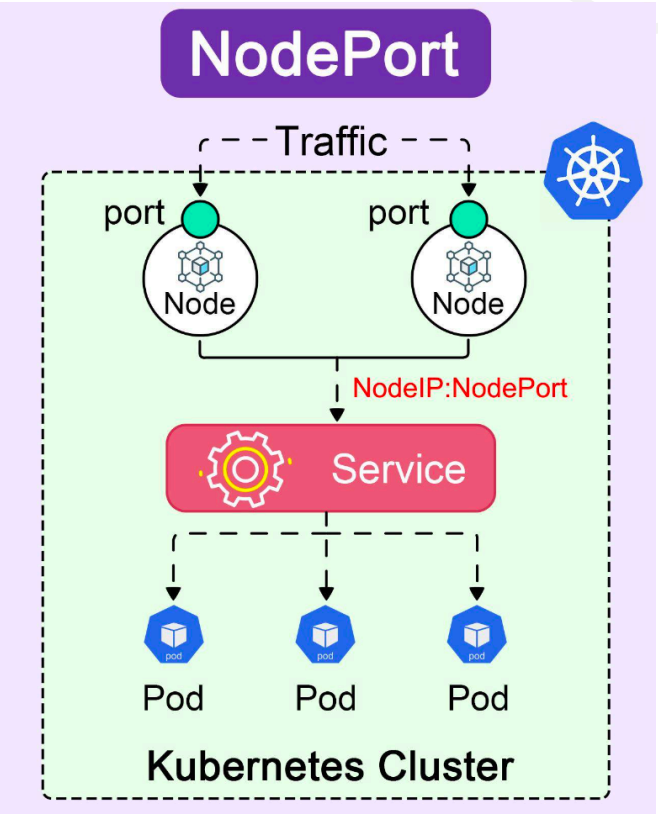
- Inter-service communication within the cluster. For example, communication between the front-end and back-end components of your app



Workshop ClusterIP Service

Service type: NodePort

- It enables **external access** to the service using the cluster nodes' **IP address** and the assigned **NodePort**
- When a NodePort service is created, Kubernetes allocates a port from a predefined range (default: **30000-32767**)



Advantages

- Provides external access to the service
- Suitable for development and testing environments

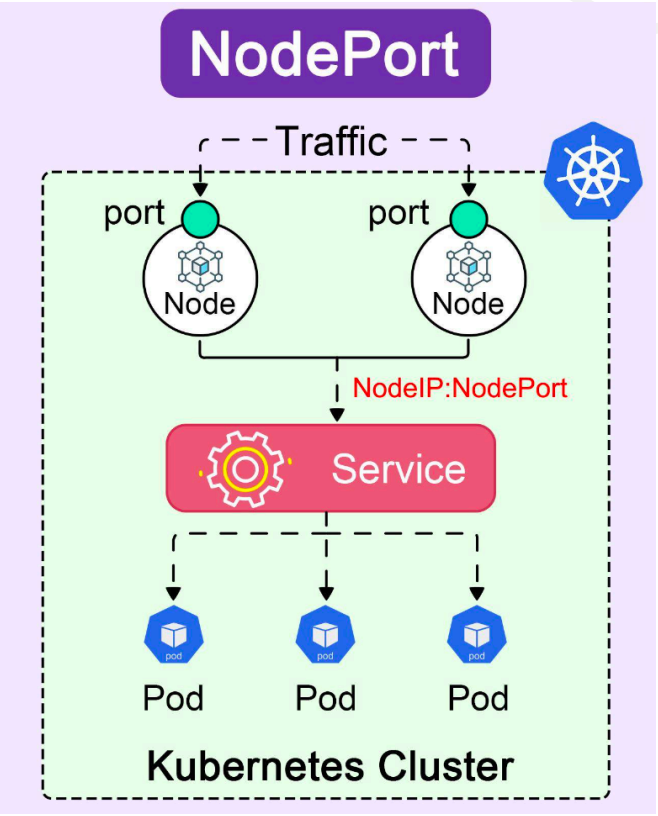
Disdvantages

- Requires manual management of port allocations
- IP Node maybe have changed
- Should not use in production

Service type: NodePort

Use case

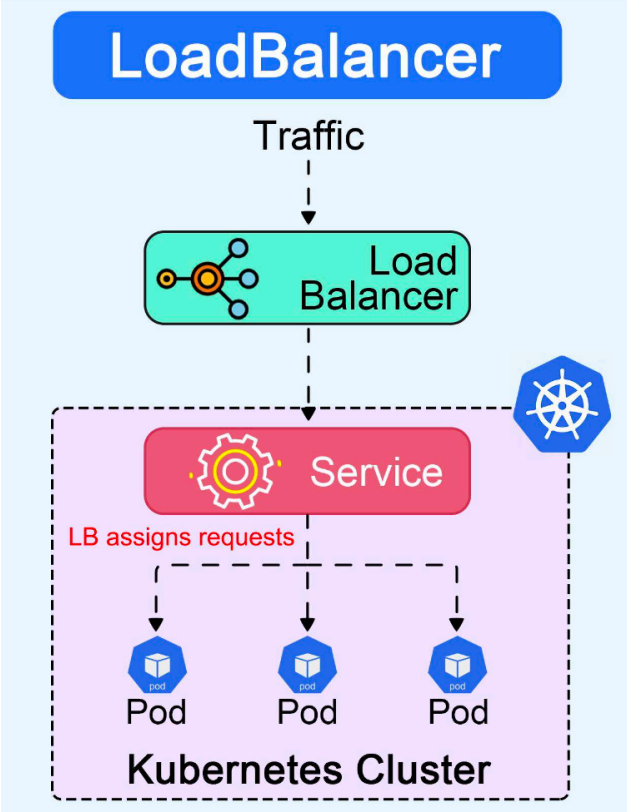
- When you want to enable external connectivity to your service
- Best for testing access for a moment time



Workshop NodePort Service

Service type: LoadBalancer

- LoadBalancer is a service type that **provisions** an **external load balancer** to distribute traffic across the pods
- Typically provided by a cloud provider's infrastructure
- When a LoadBalancer service is created, Kubernetes interacts with the cloud provider's API to an external load balancer



Advantages

- Provides external access to the service
- Automatically provisions an external load balancer for the service
- Supports scaling and high availability

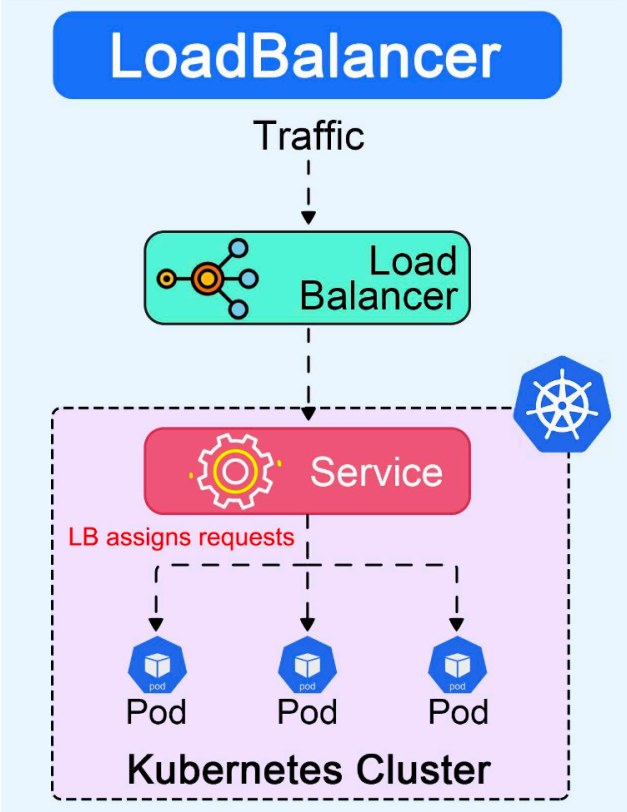
Disdvantages

- Requires a cloud provider
- Additional costs for using the load balancer service on cloud provider

Service type: LoadBalancer

Use case

- When you want to enable external connectivity to your service
- When you are using a cloud provider to host your Kubernetes cluster



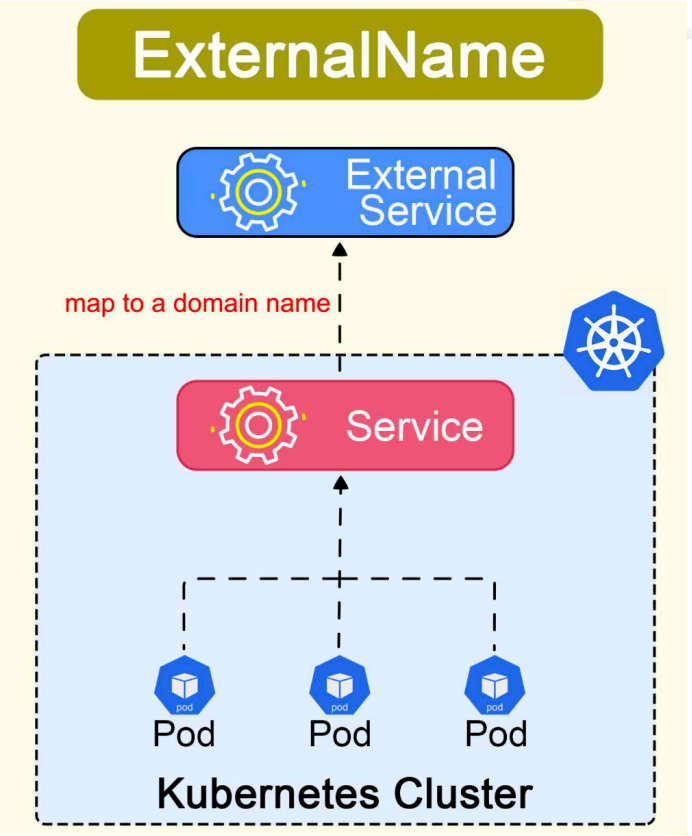
Workshop LoadBalancer Service

Service type: ExternalName

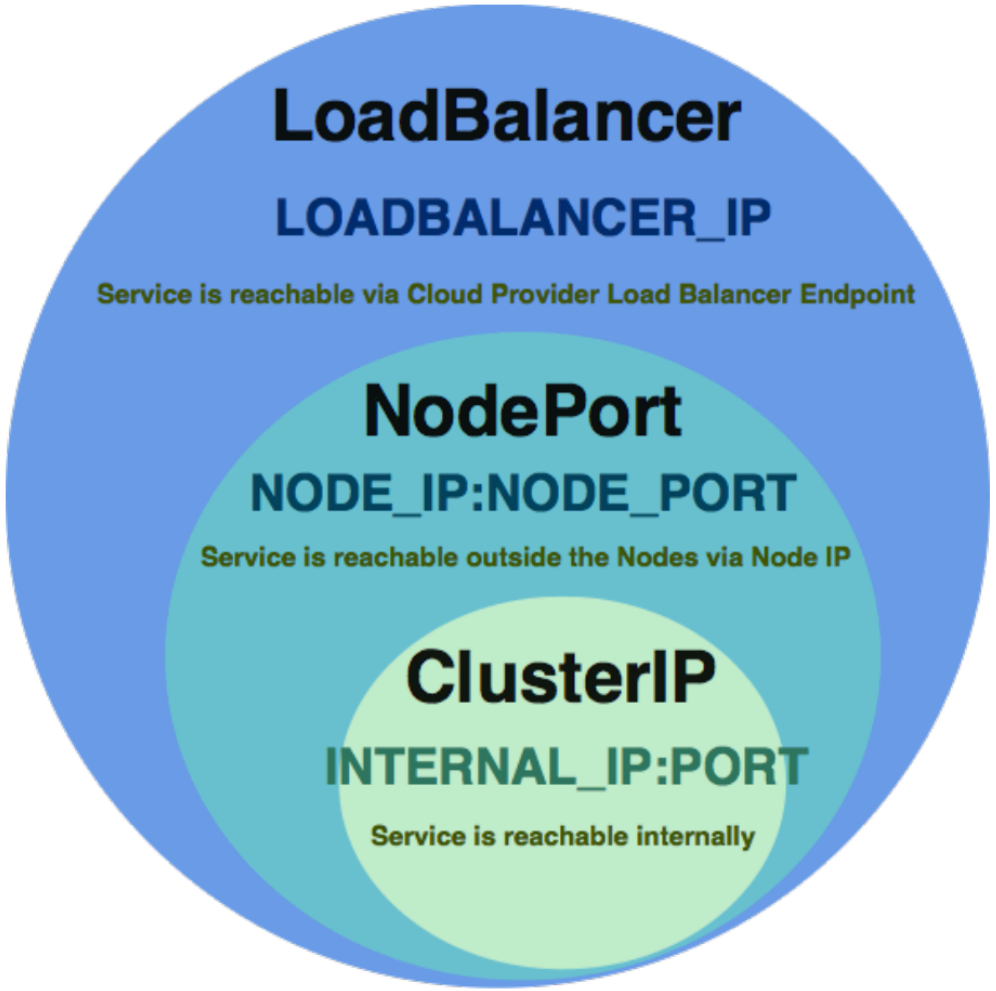
- An ExternalName Service is a special type of Kubernetes service that allows you to expose an existing external service (like a database or a third-party API) to your cluster without deploying any Pods

Use cases

- Accessing an external database to your Kubernetes cluster
- Connecting a third-party API from within your cluster



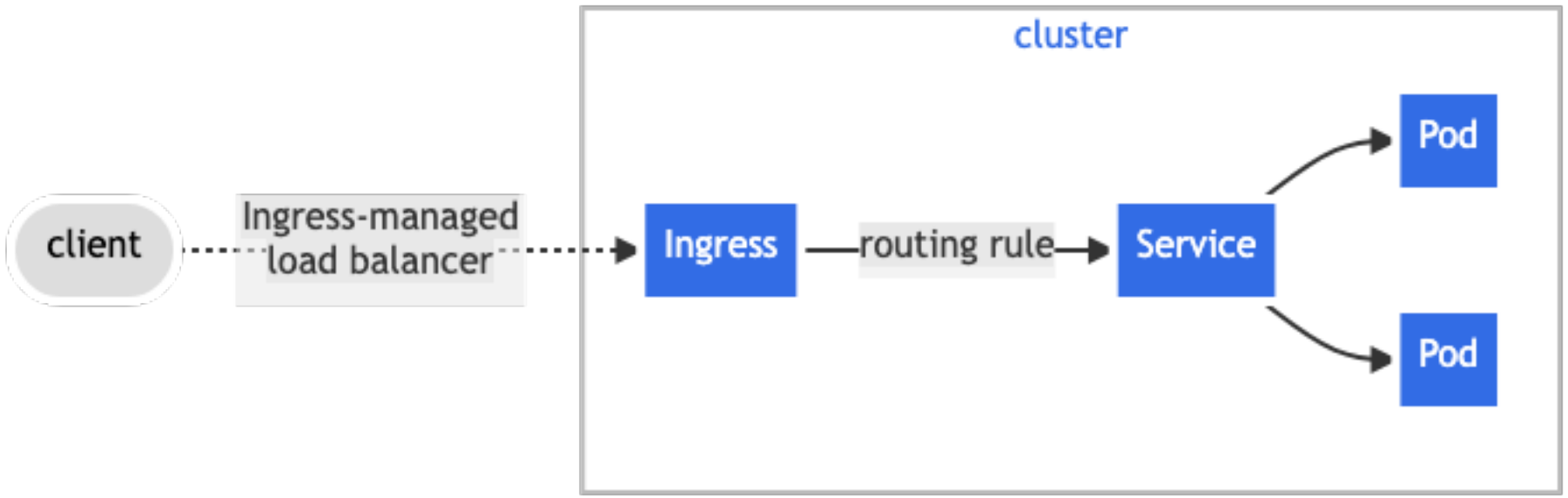
Service type



Ingress

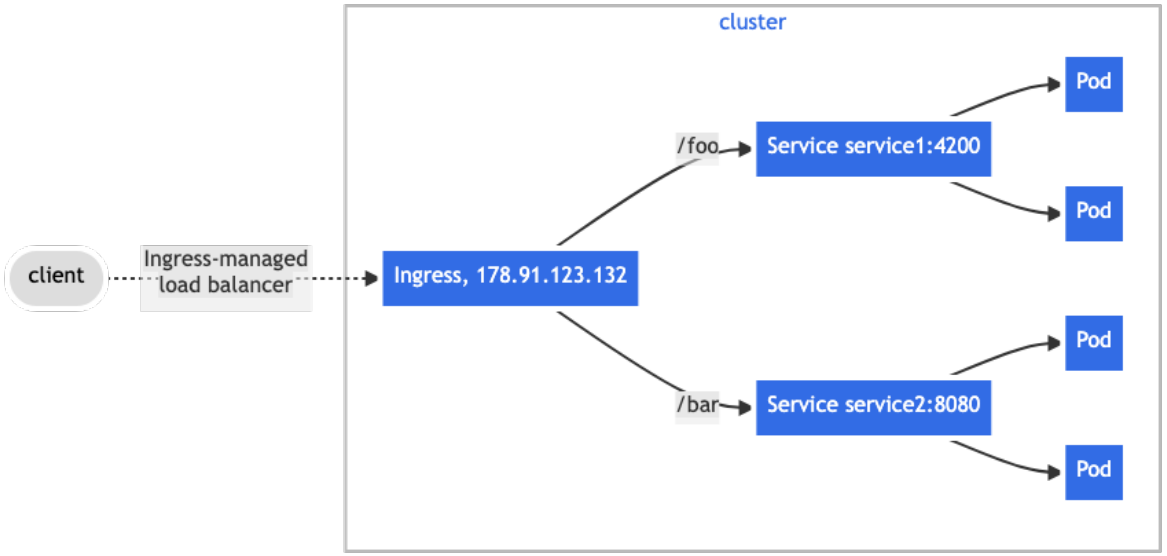
Ingress

- Routing from outside the cluster to services within the cluster
- Traffic routing is controlled by rules defined on the ingress resource
- An **Ingress controller** is responsible for fulfilling the Ingress



Ingress Controllers

- The Ingress Controller is an application hosted inside a Kubernetes cluster that actively **manages traffic** following Ingress Resources and pre-defined Ingress Rules
- Ingress controllers doesn't come with standard, have to be deployed separately
- Popular controllers include ingress-nginx, HAProxy, Traefik

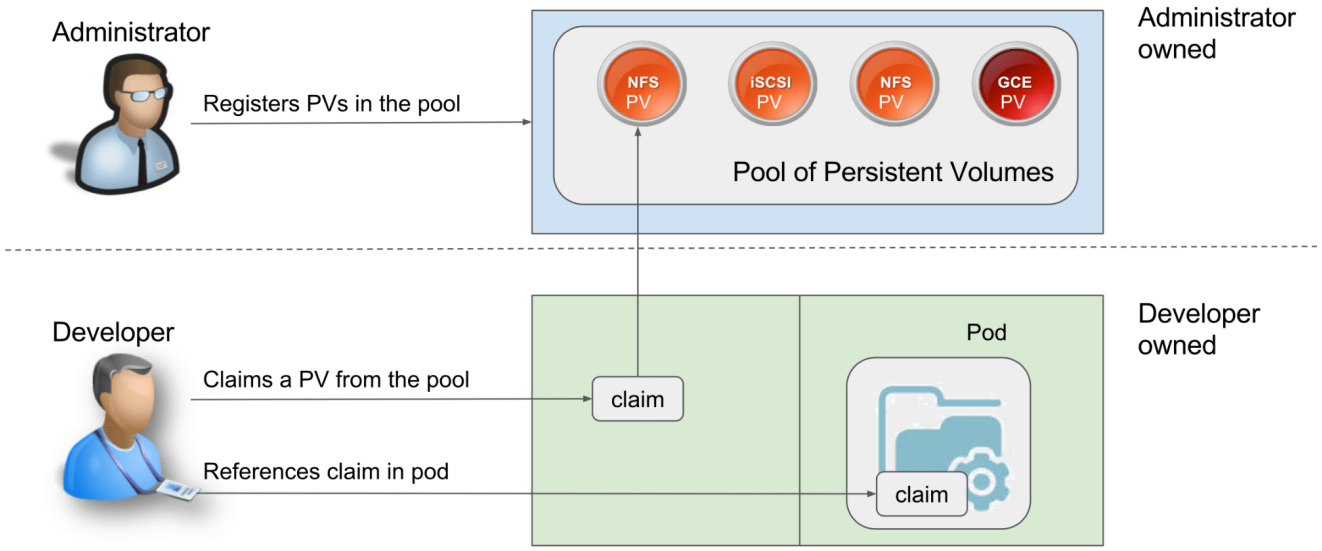


Workshop Ingress

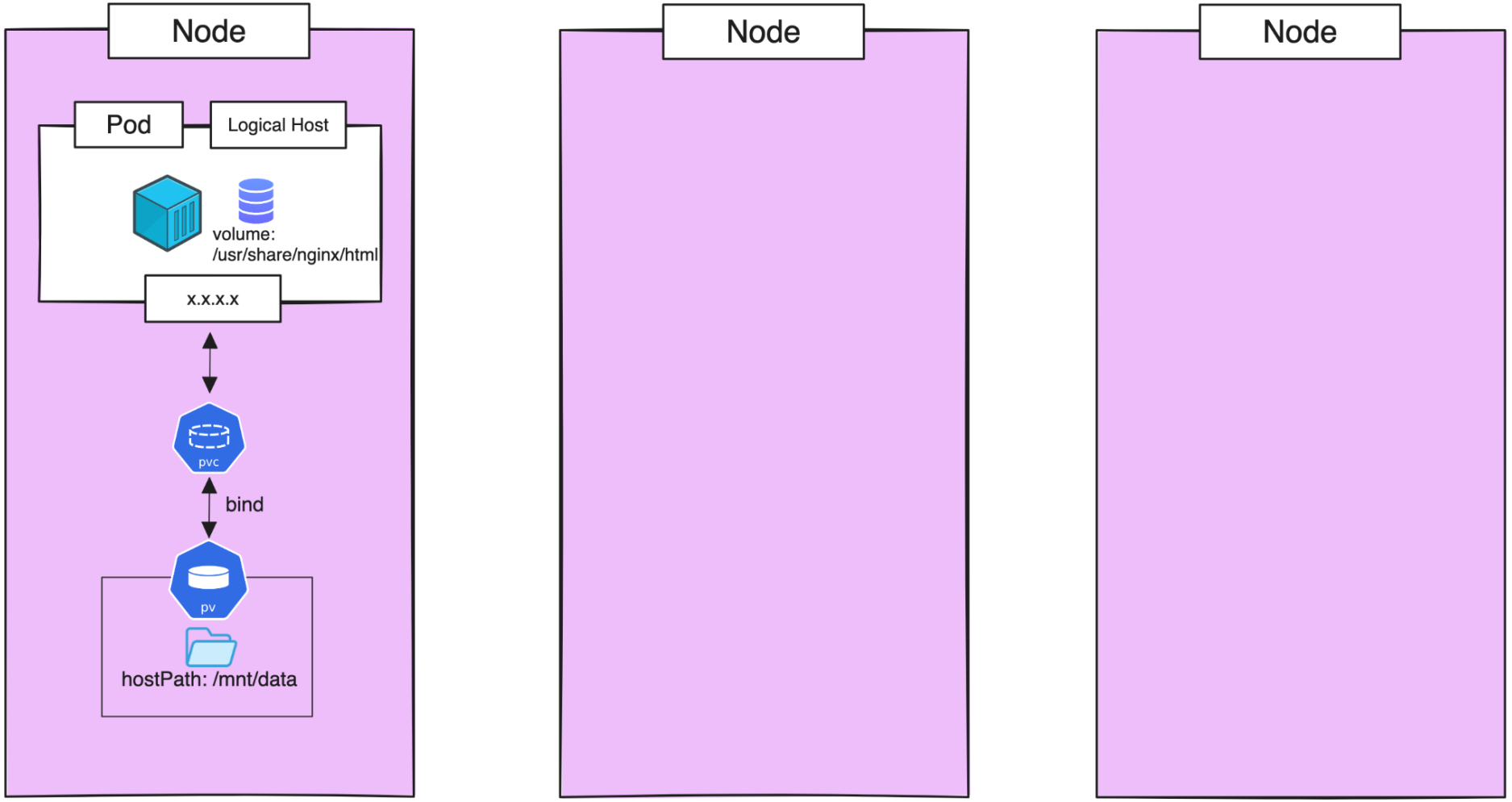
Persistent Volume

Persistent Volume

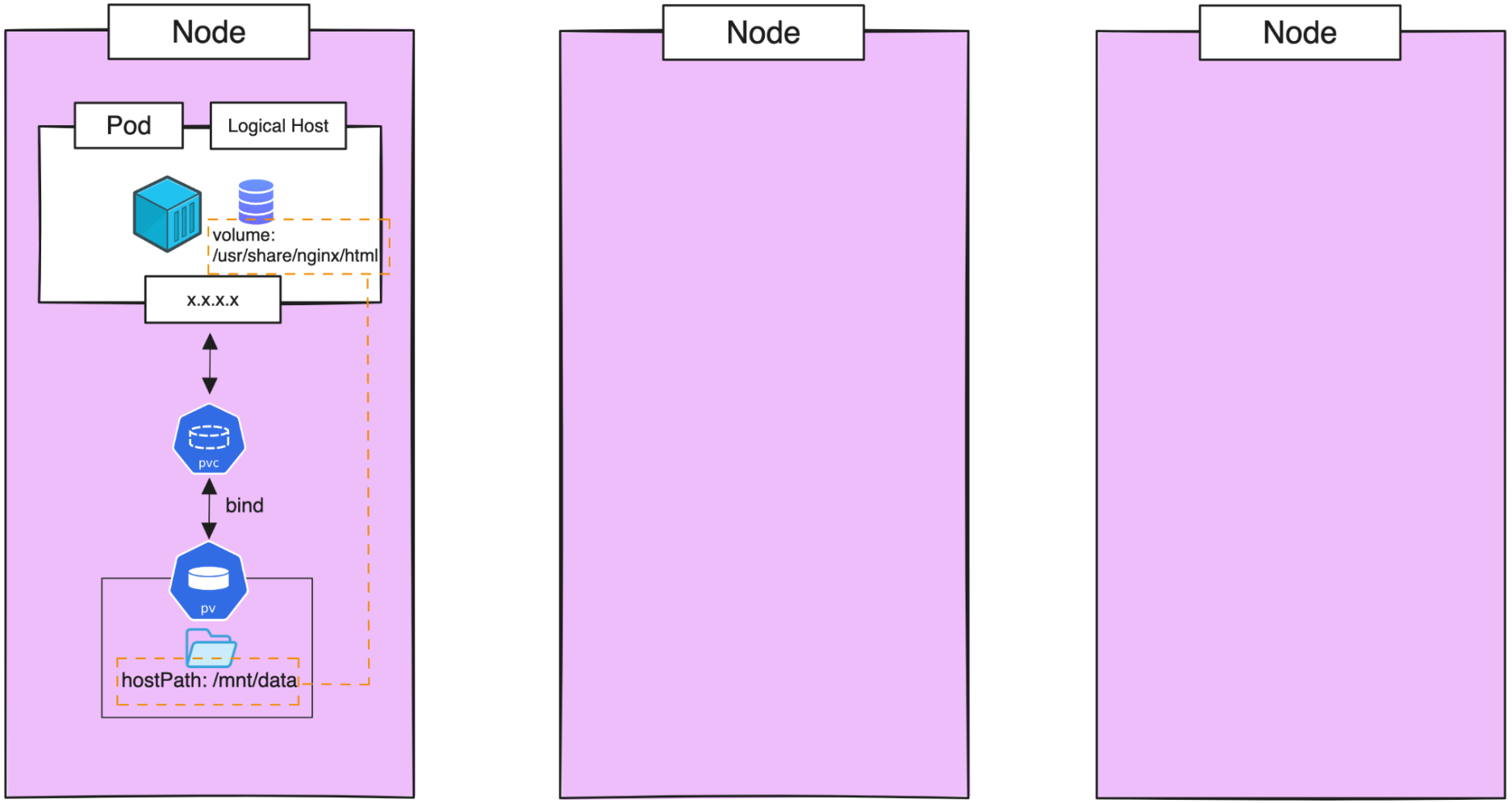
- Persistent Volume (PV) is a piece of storage in the cluster
- Can be provisioned from various sources like host local storage, cloud providers storage, or NFS.
- Persistent Volume Claims (PVC) allow a user to consume abstract storage resources
- A PVC to PV binding is a one-to-one mapping



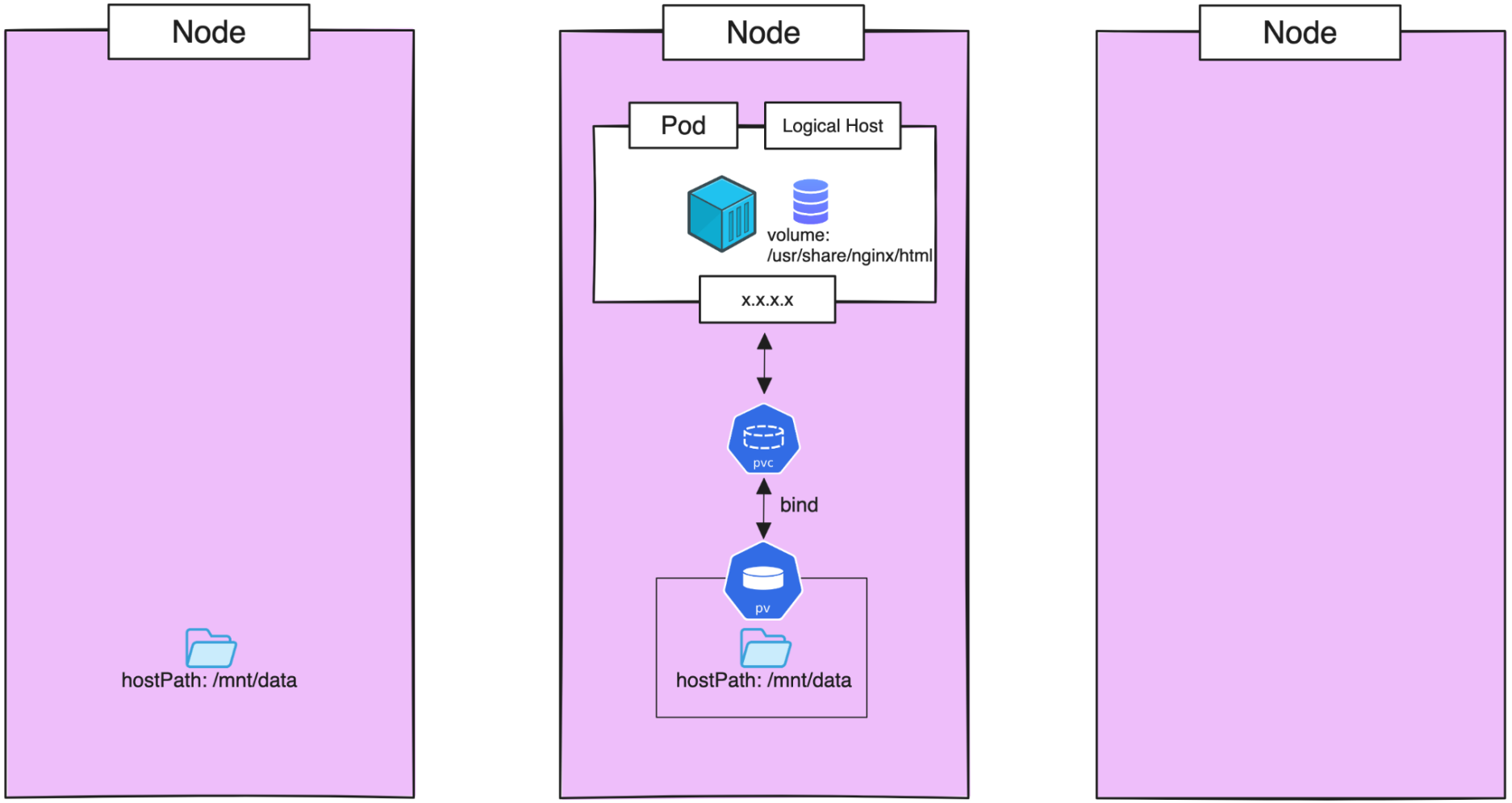
PV & PVC



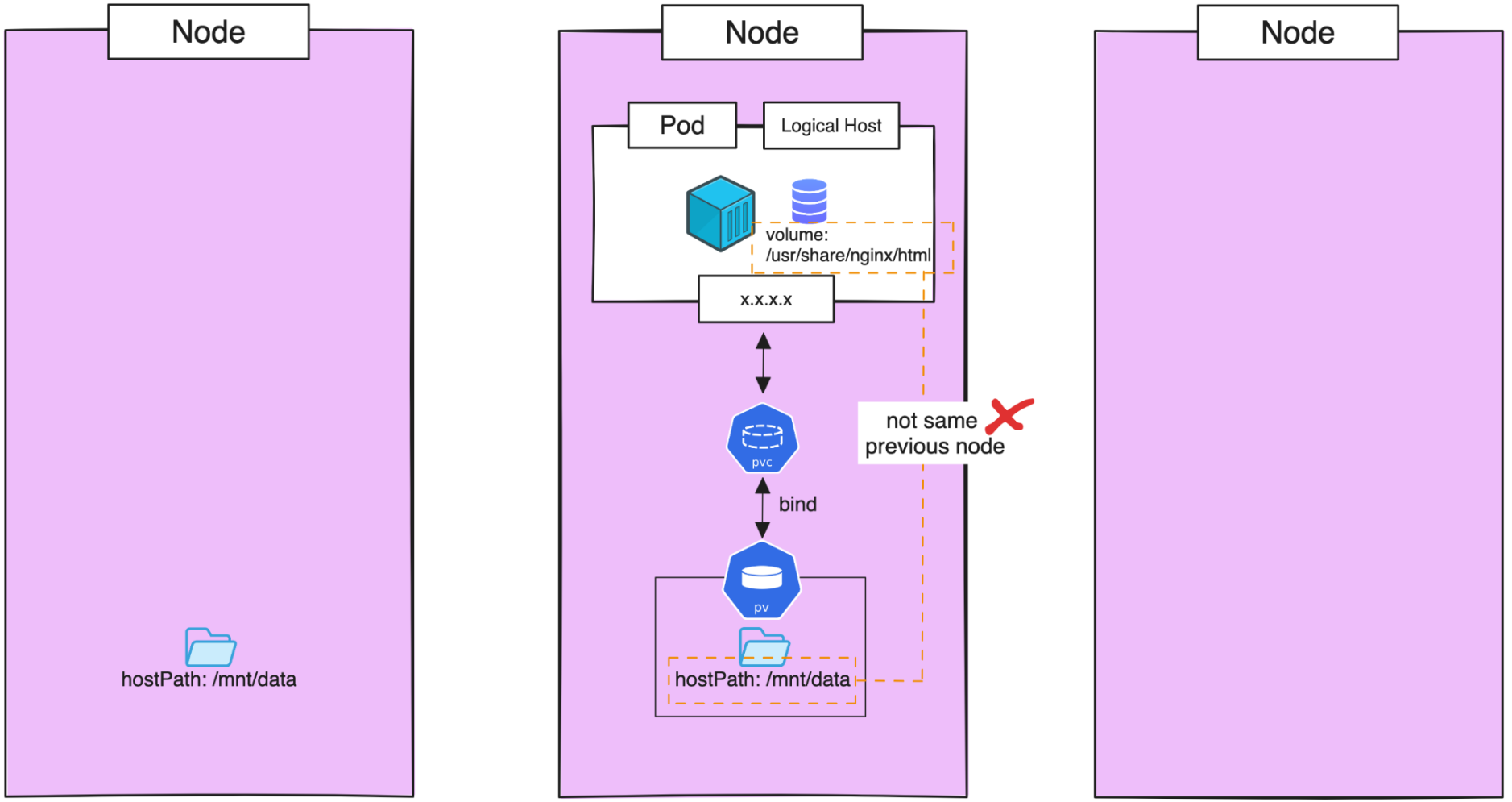
PV & PVC



PV & PVC



PV & PVC



PV & PVC

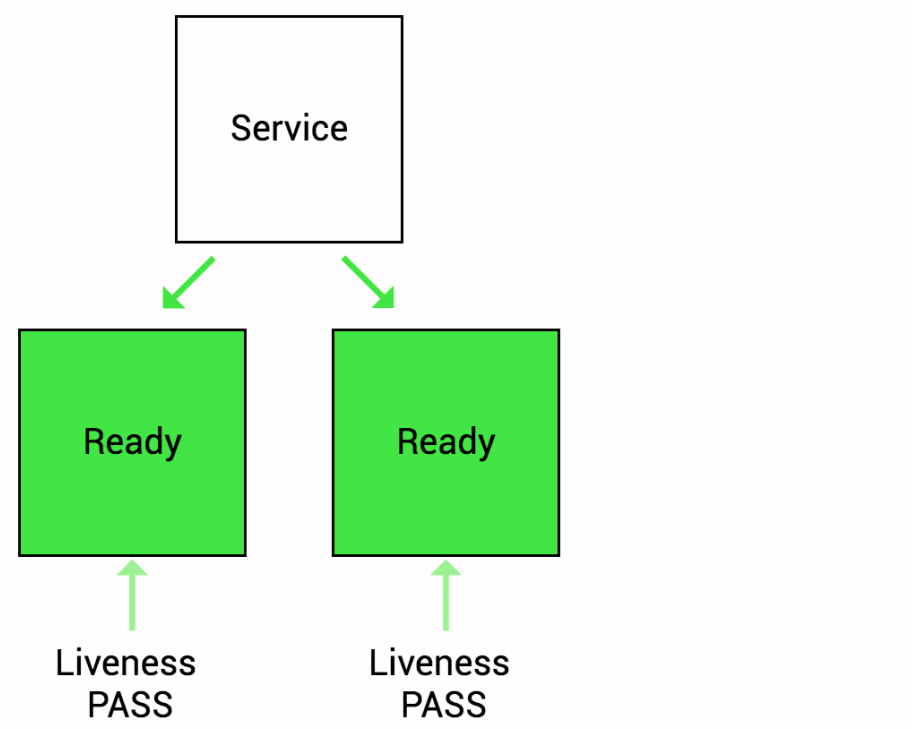
Solutions

- Using selector node or nodeAffinity for ensure the target node is same
- Using shared storages like NFS or cloud storages

Pod liveness & readiness

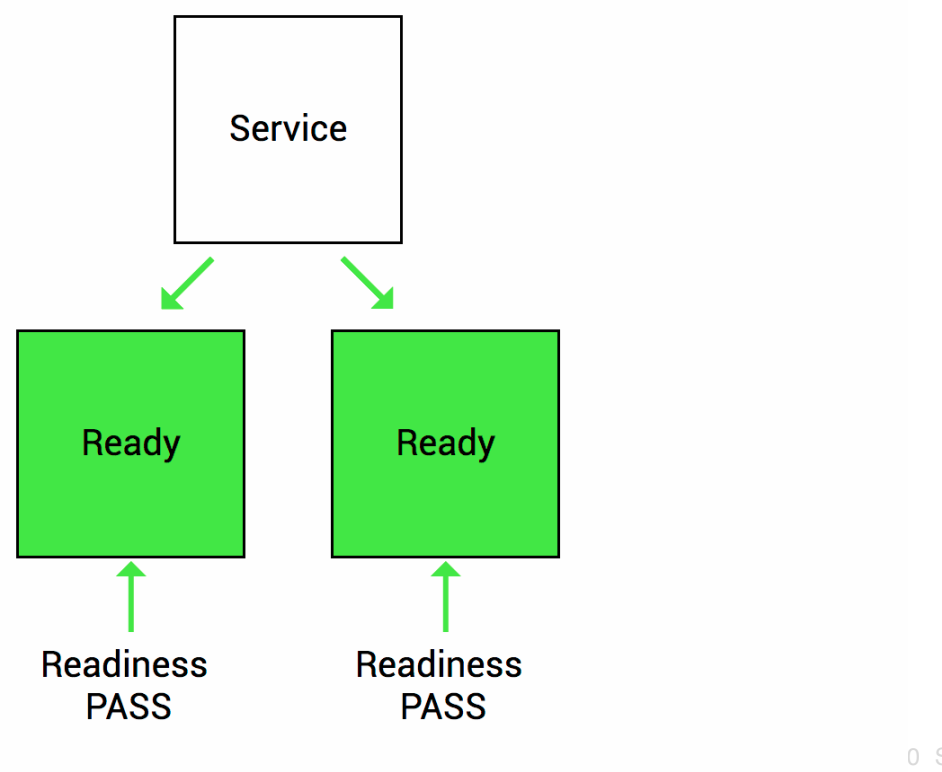
Pod liveness & readiness

Liveness: Indicates whether the Container is running. and if for some reason the liveness probe fails, it restarts the container



Pod liveness & readiness

Readiness: Indicates whether the Container is ready to service requests. If the condition inside readiness probe passes, only then our application can serve traffic.

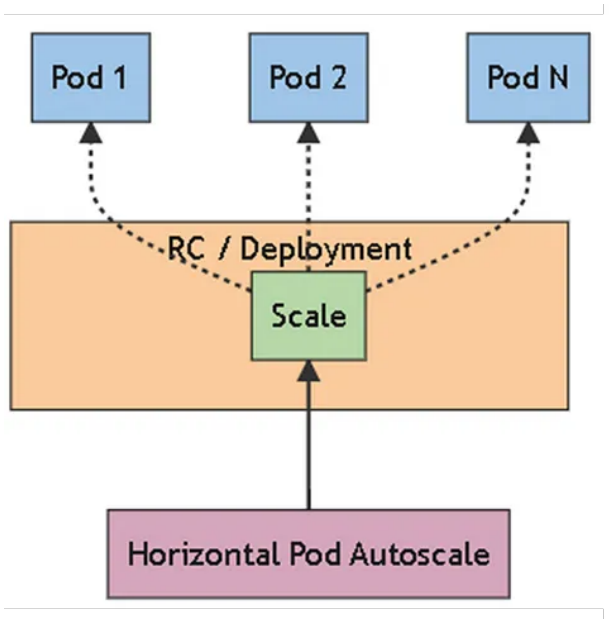


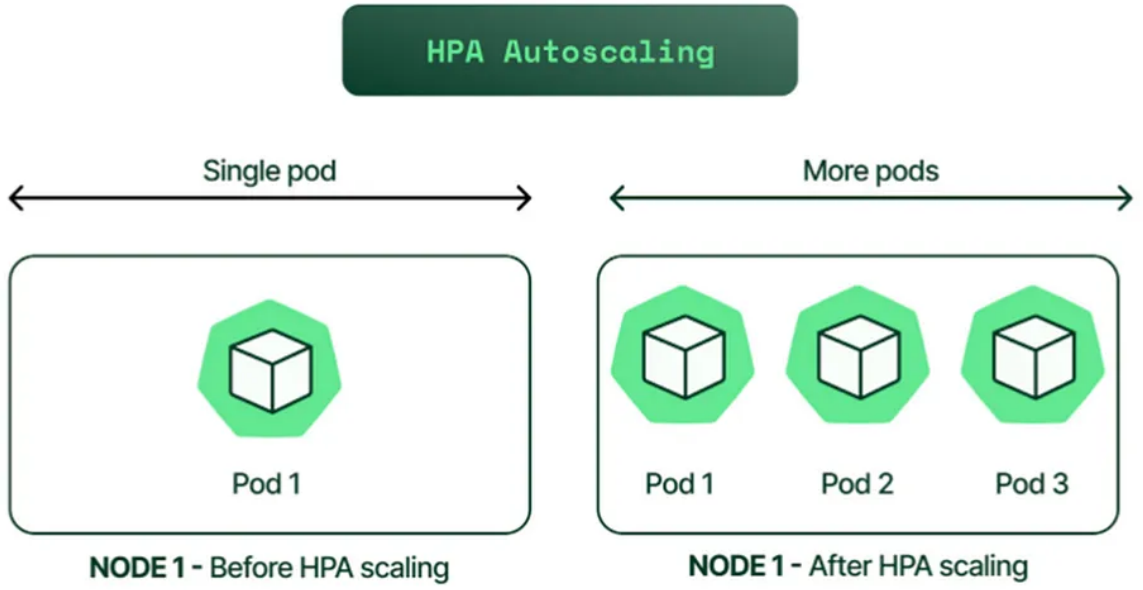
Workshop liveness & readiness

HPA (Horizontal Pod Autoscaler)

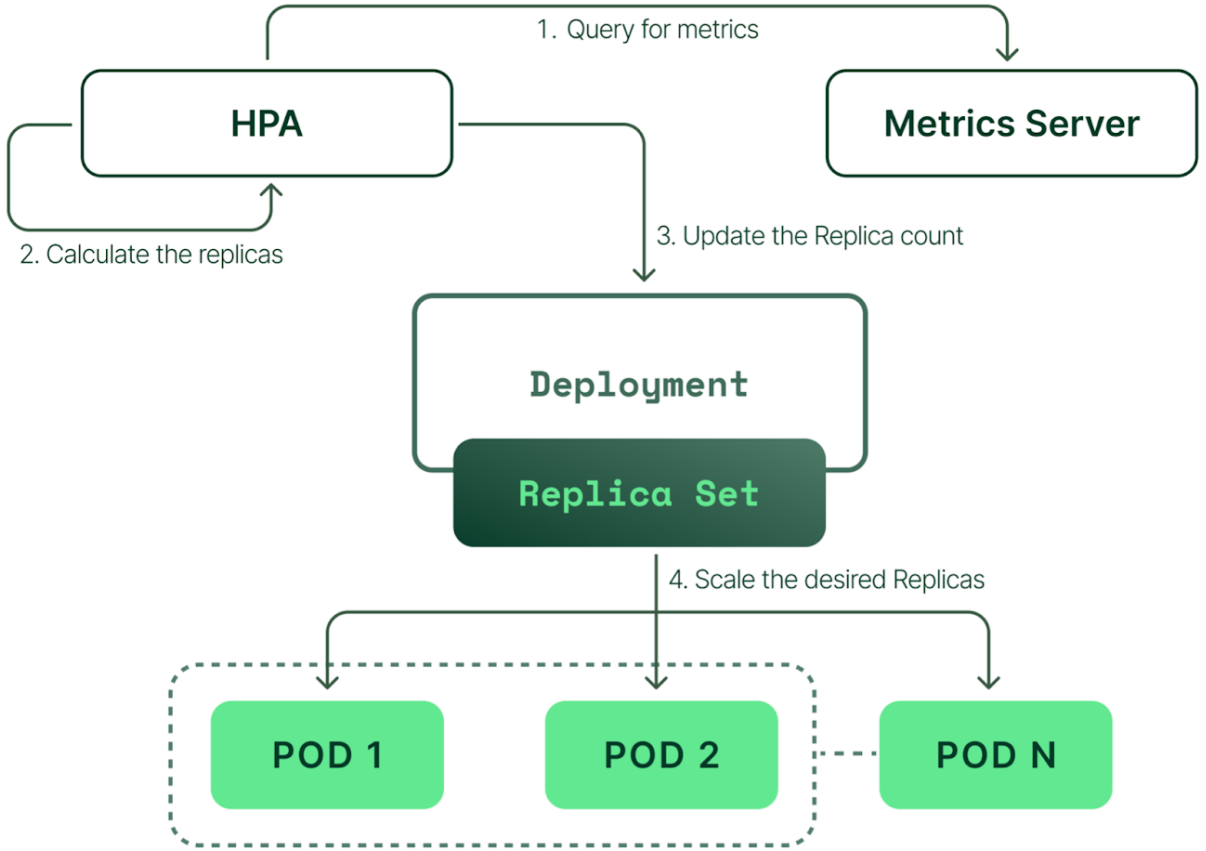
HPA

- A HorizontalPodAutoscaler (HPA for short) automatically updates a workload resource, with the aim of automatically scaling the workload to match demand





How does HPA work?



Workshop HPA

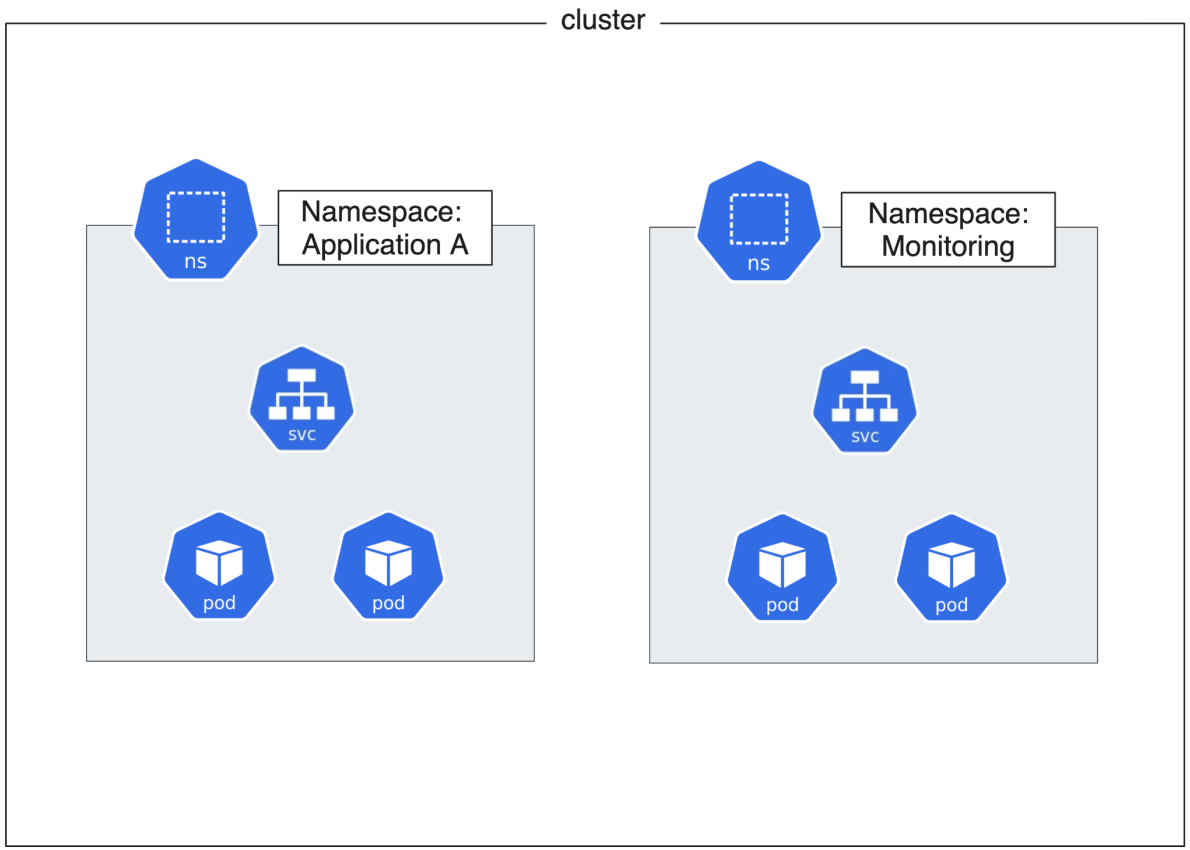
Namespace

Namespace

- Partition a single kubernetes cluster into multiples clusters
- 4 defaults namespaces
  - **kube-system**: core system processes
  - **kube-public**: public accessible data, ex: ConfigMaps & Secrets
  - **kube-node-lease**: heartbeat of nodes, so that the control plane can detect node failure
  - **default**: your resources/objects

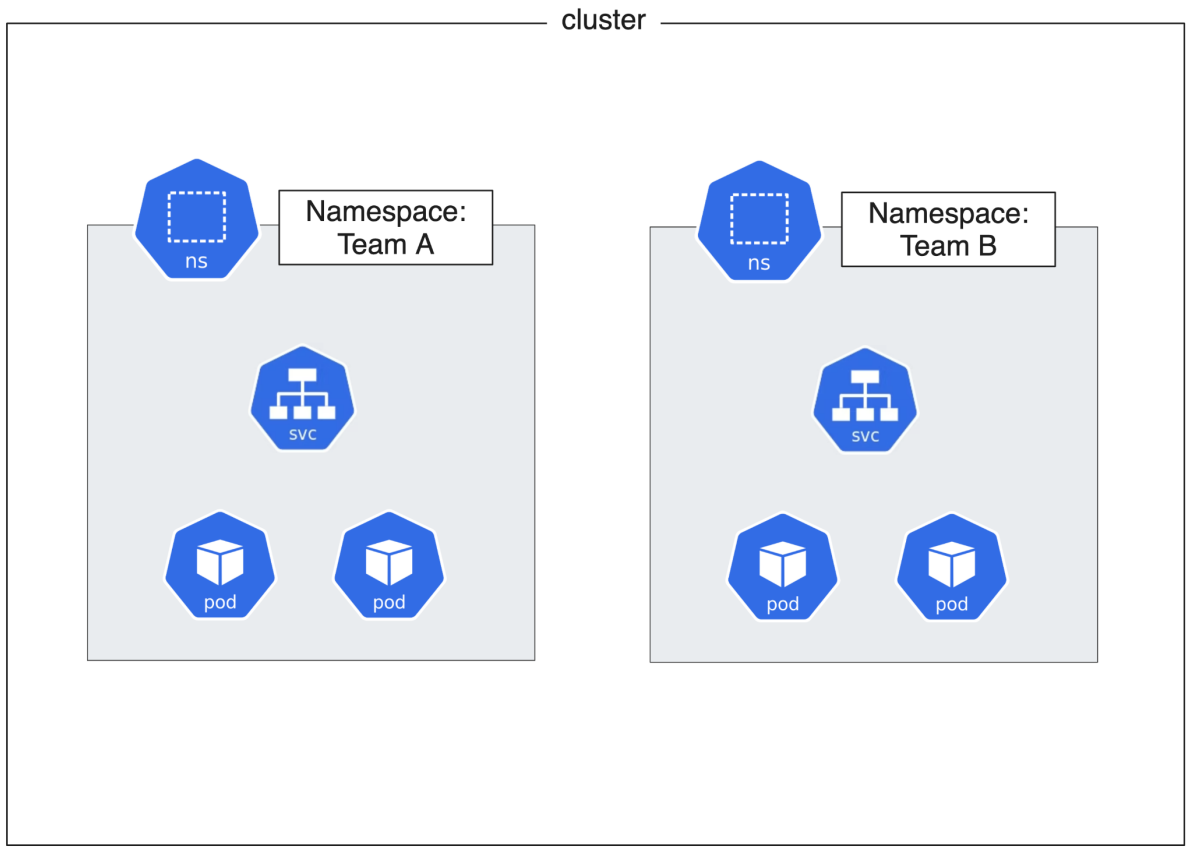
Namespace use cases

- Application domain



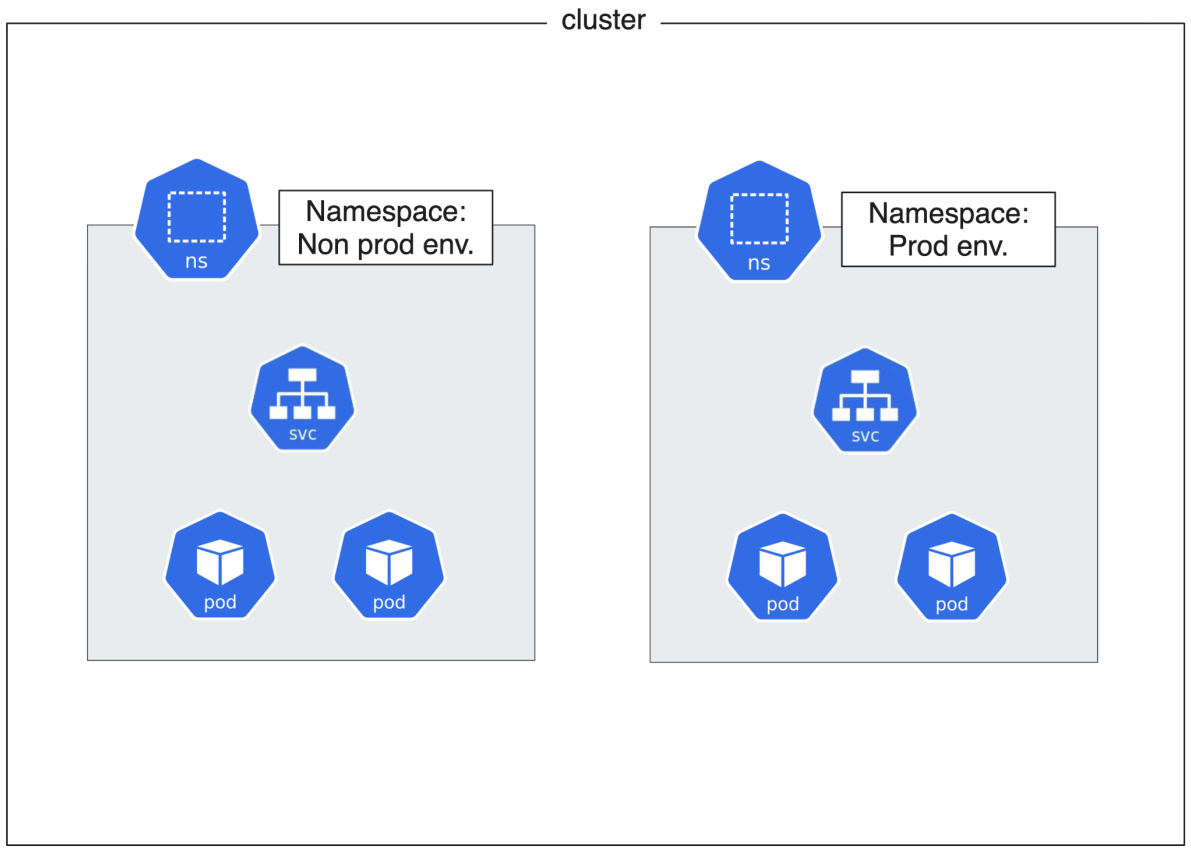
Namespace use cases

- Teams driven



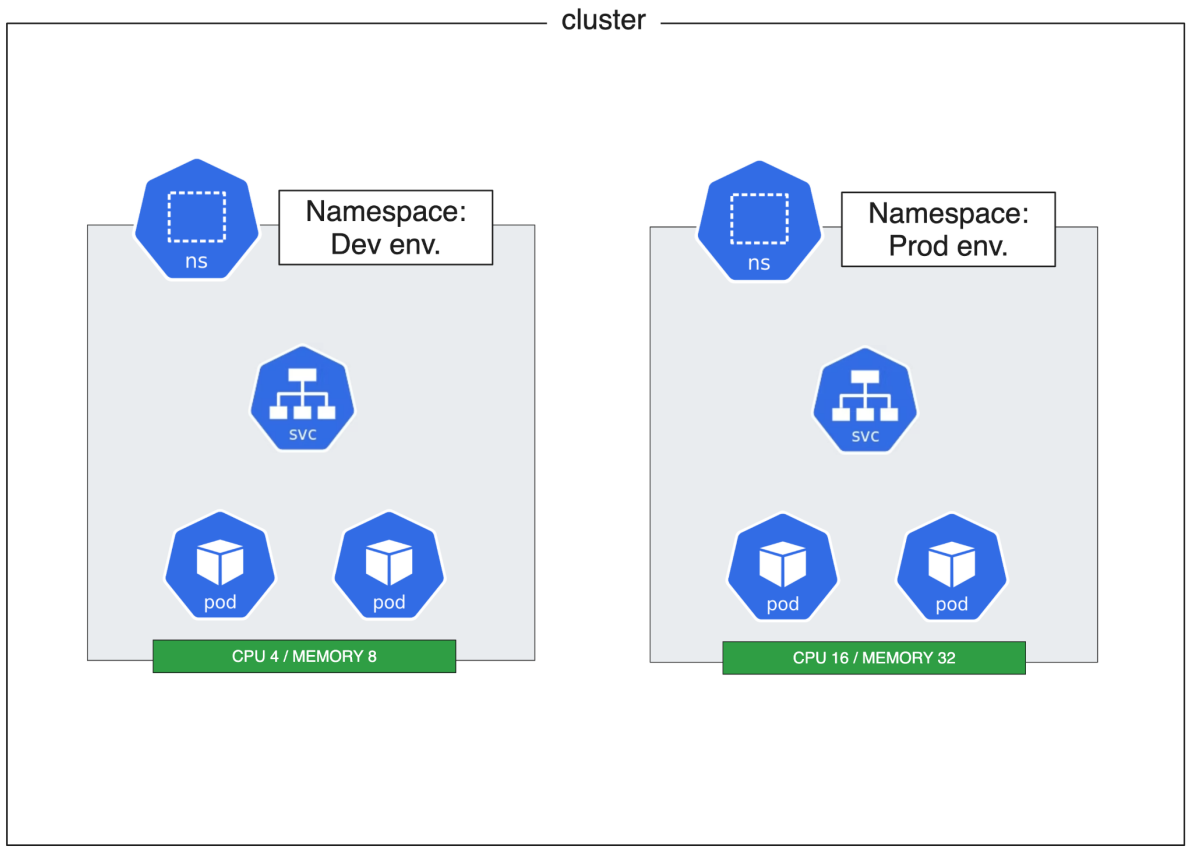
Namespace use cases

- Environments



Namespace use cases

- Resources quota limit



HELM

HELM

HELM is the package manager for Kubernetes

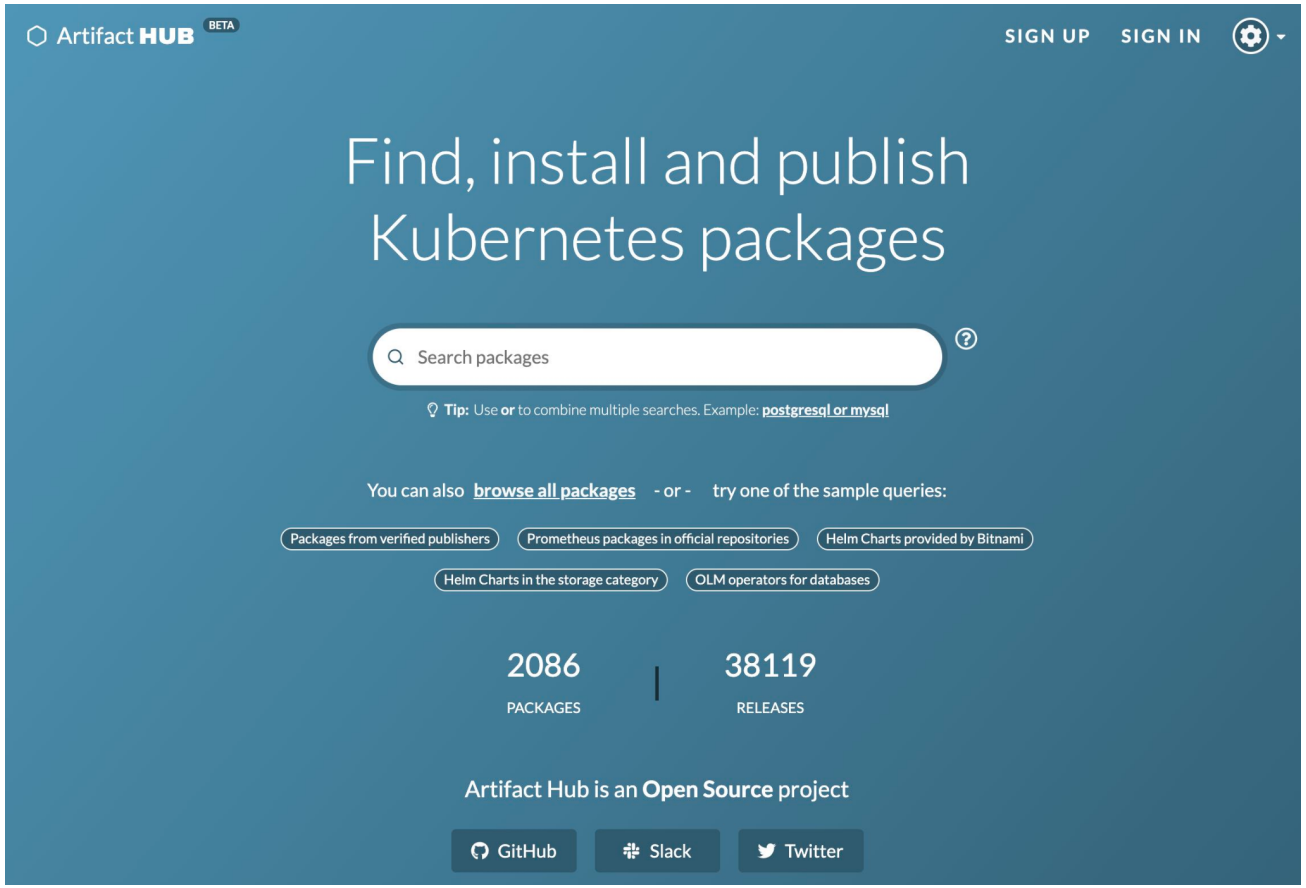
HELM

HELM is the package manager for Kubernetes

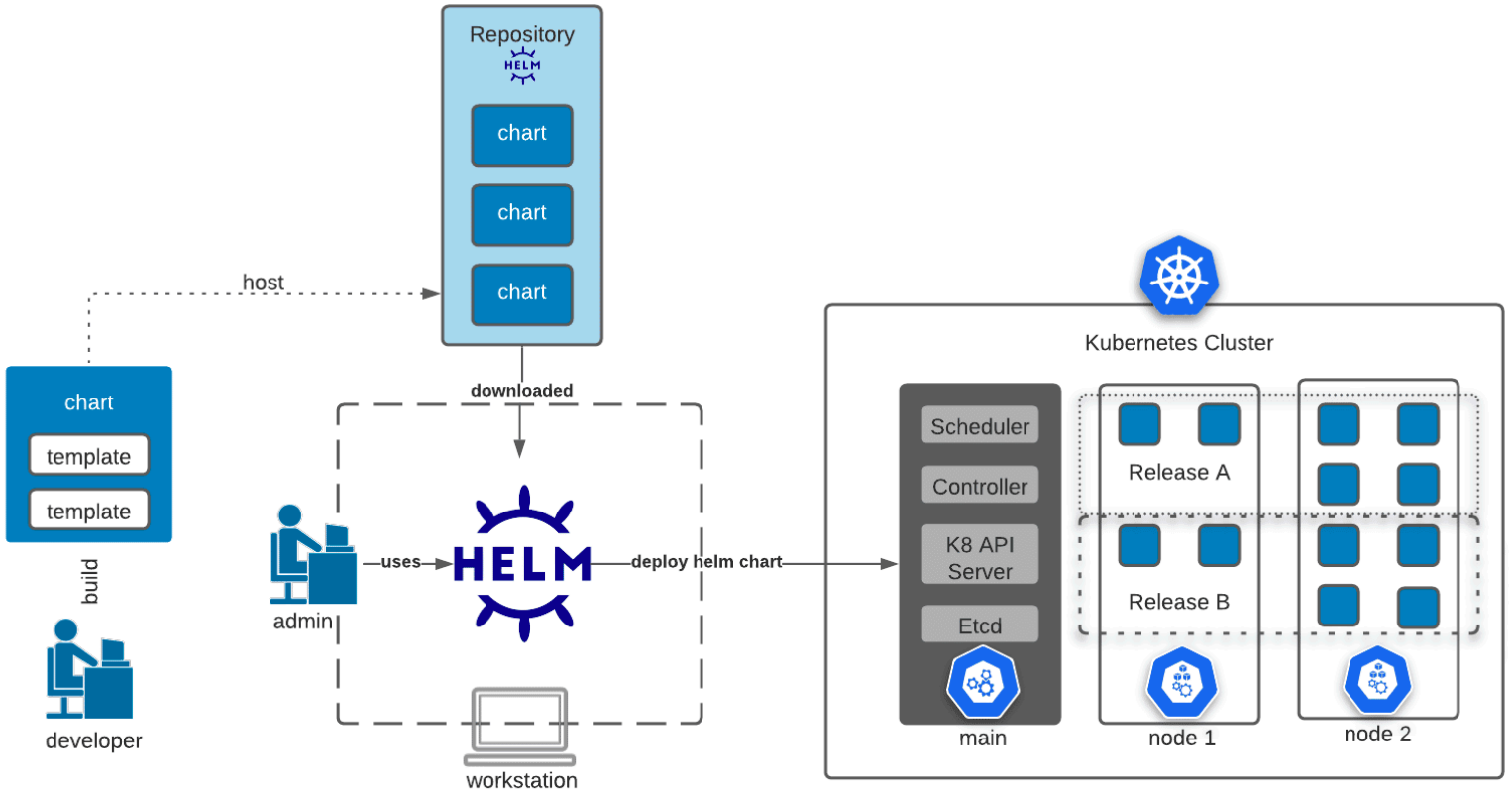
- Boosts productivity
- Reduces complexity
- Simple sharing
- Versioning and Upgrades

HELM Charts

Public HELM Chart: https://artifacthub.io/

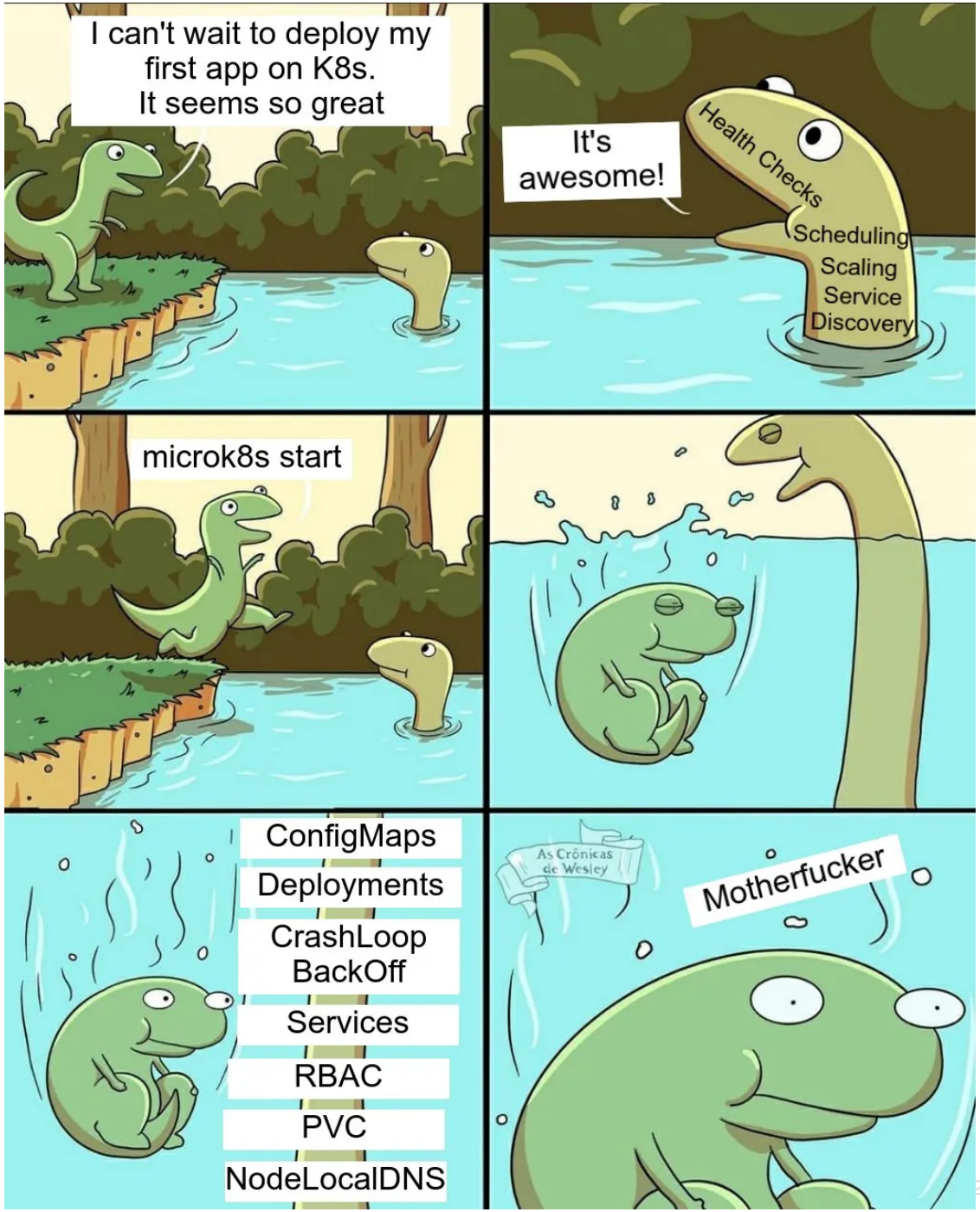


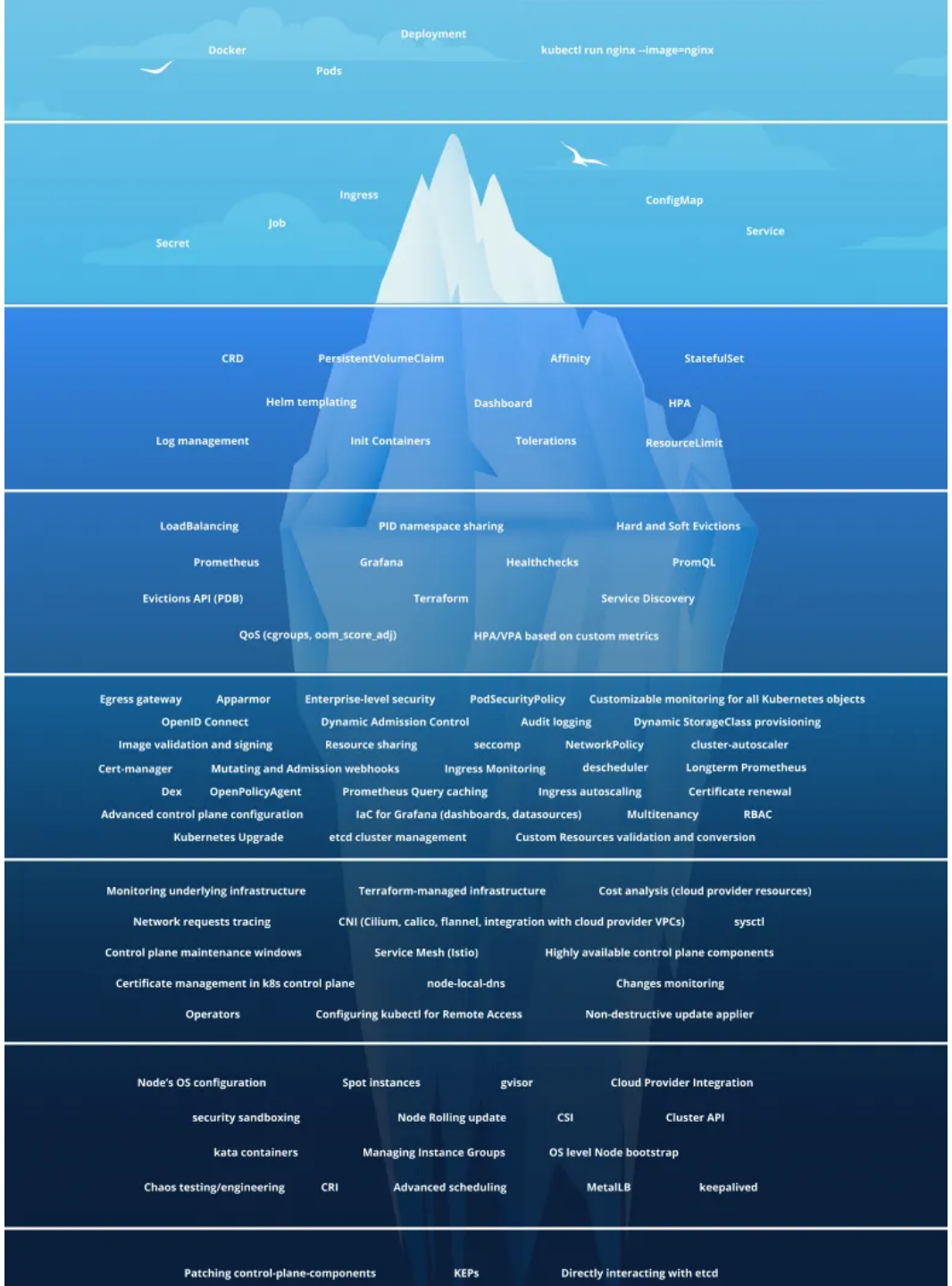
HELM Charts

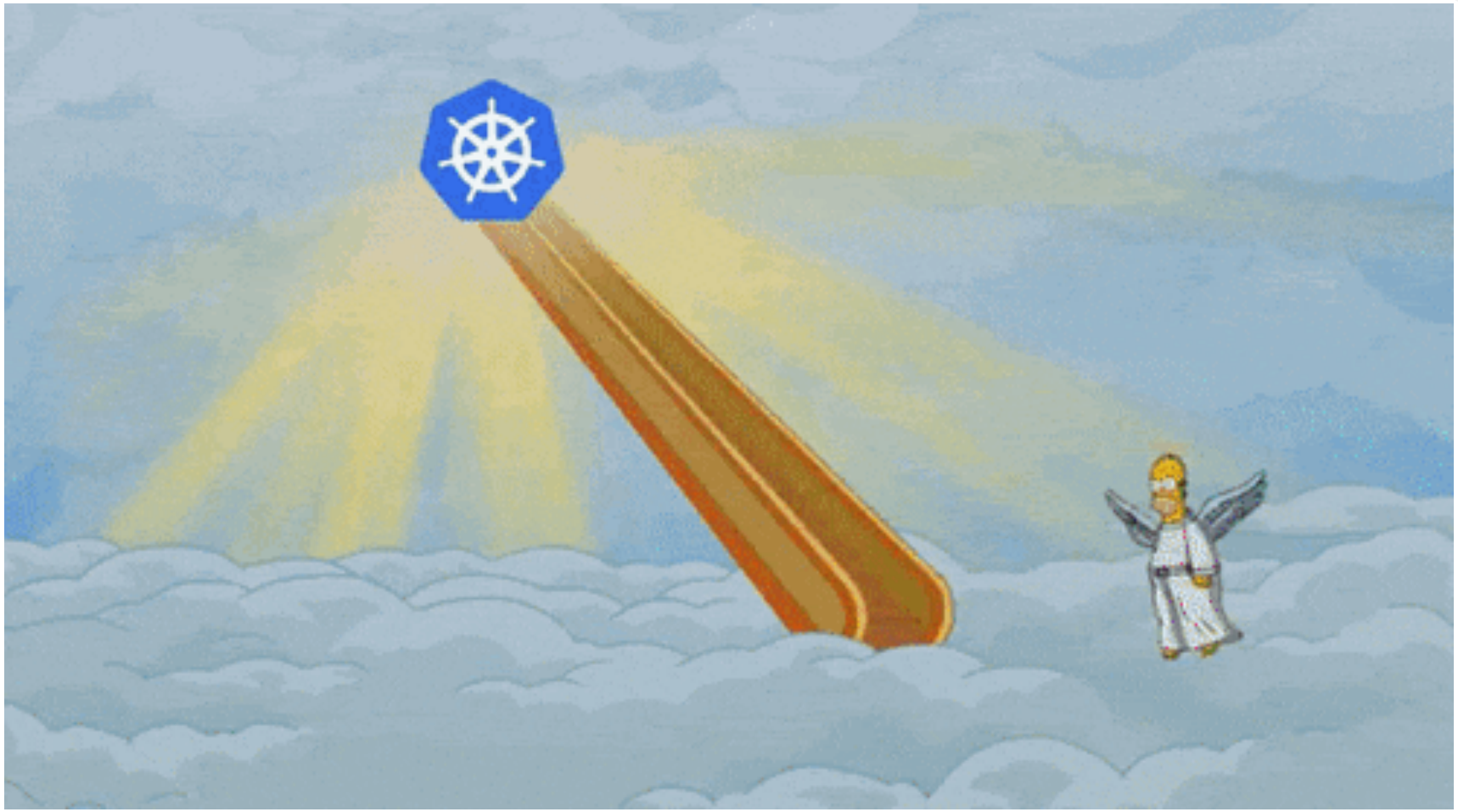


Helm Workflow

Workshop HELM







DevOps

Agenda

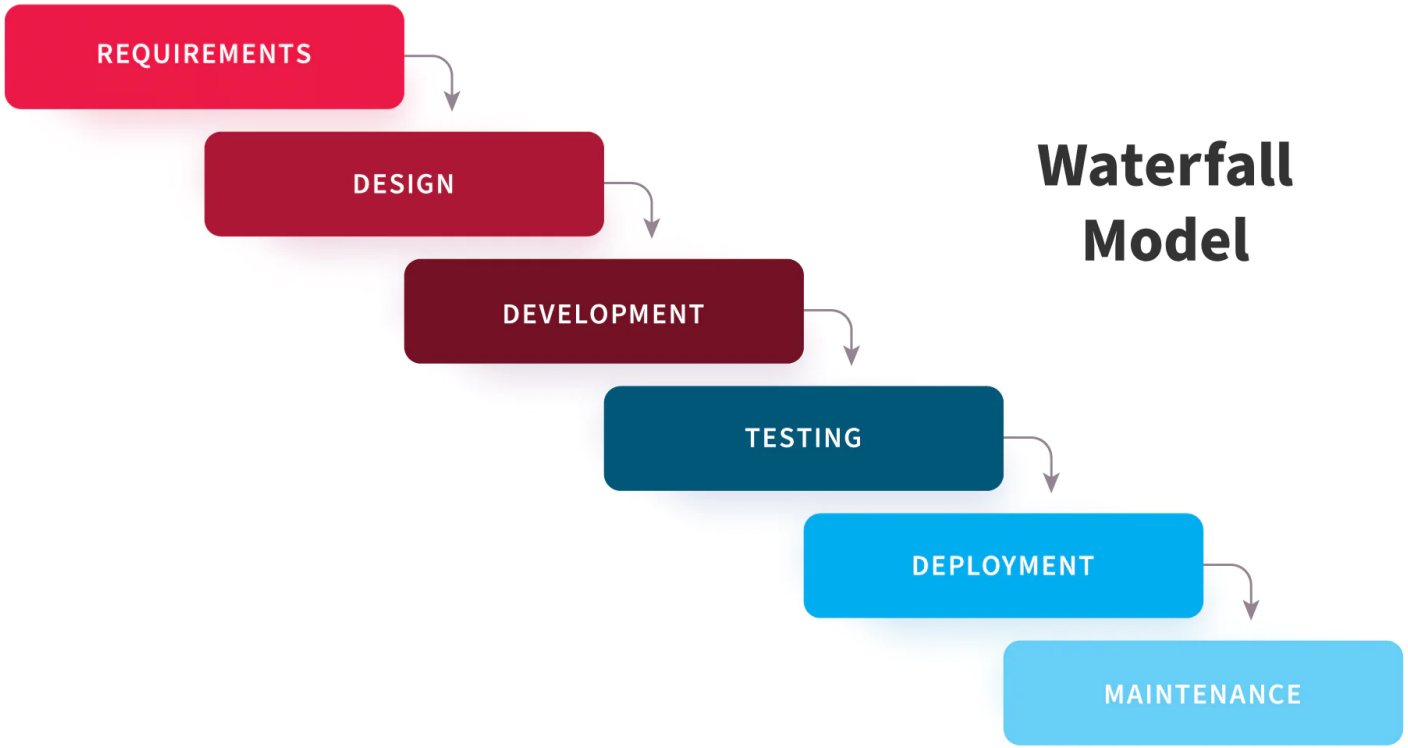
- Introduction
- What is DevOps ?
- How to start DevOps ?
- CI/CD
  - Continuous integration
  - Continuous deployment
- Deployment strategy
- Observability

Introduction

Before DevOps

- In the early days, software was developed and maintained by a small group of specialists
- The development process known as “Waterfall”
- Spending months or even years building and testing software before it was ready to be deployed
- Breaks project activities into phases, each phase must be completed before the next phase can begin

Waterfall Model



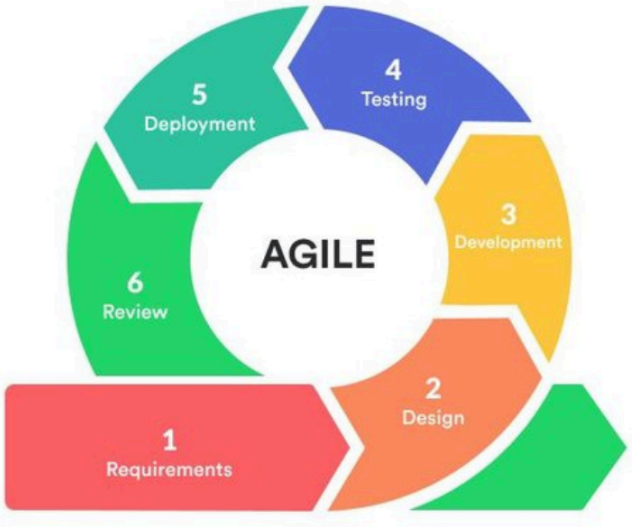
Waterfall Model

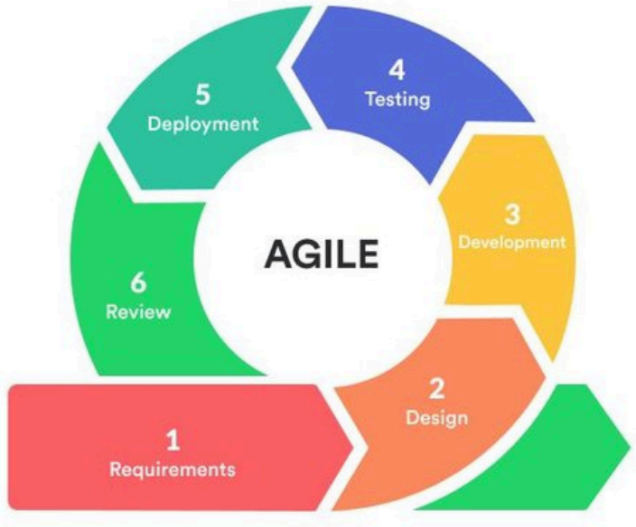
- Lack of flexibility
- No room for errors
- Difficulty in dealing with changes
- No customer involvement

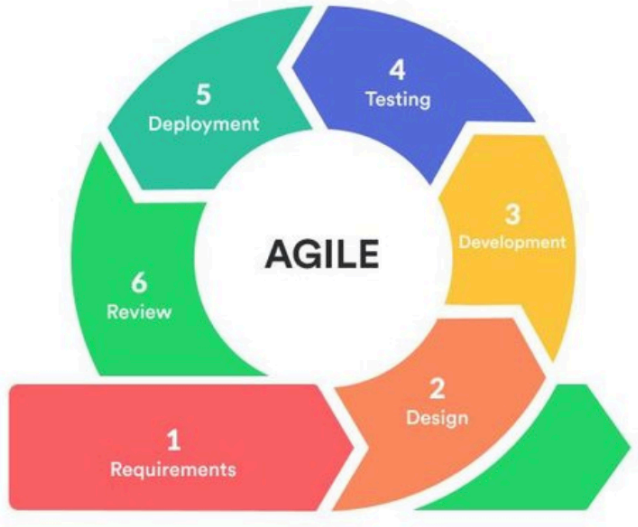
The rise of the Agile methodology

- Focusing on the clean delivery of individual pieces or parts, not the entire application
- Get new features into the hands of the users quickly
- Focusing on rapid iteration
- Emphasizing collaboration, flexibility and continuous improvement

Agile





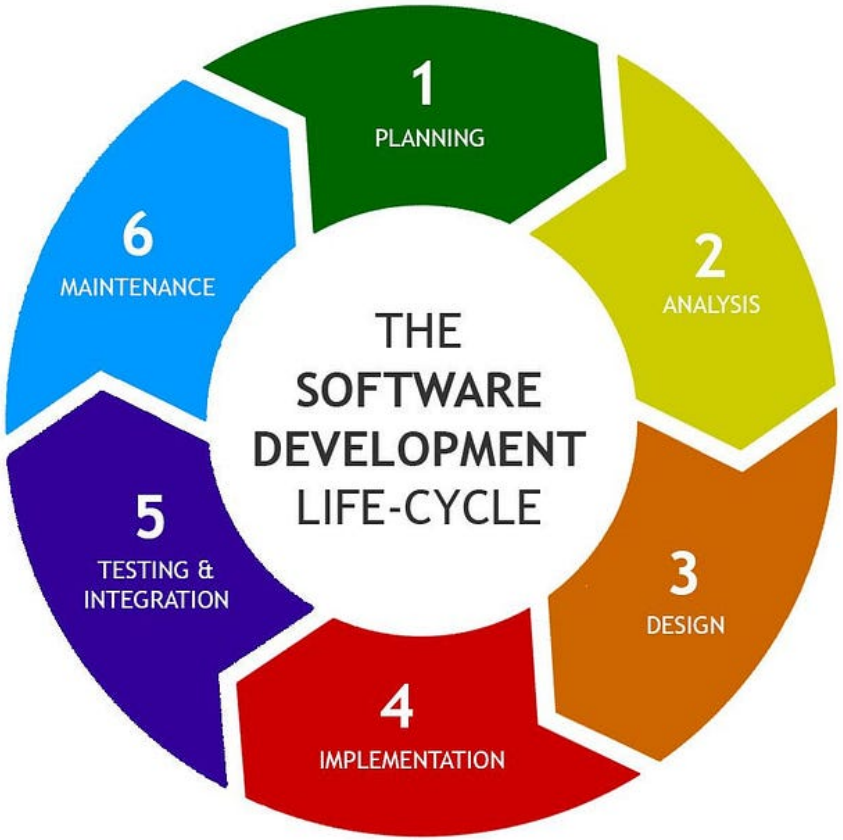


Agile

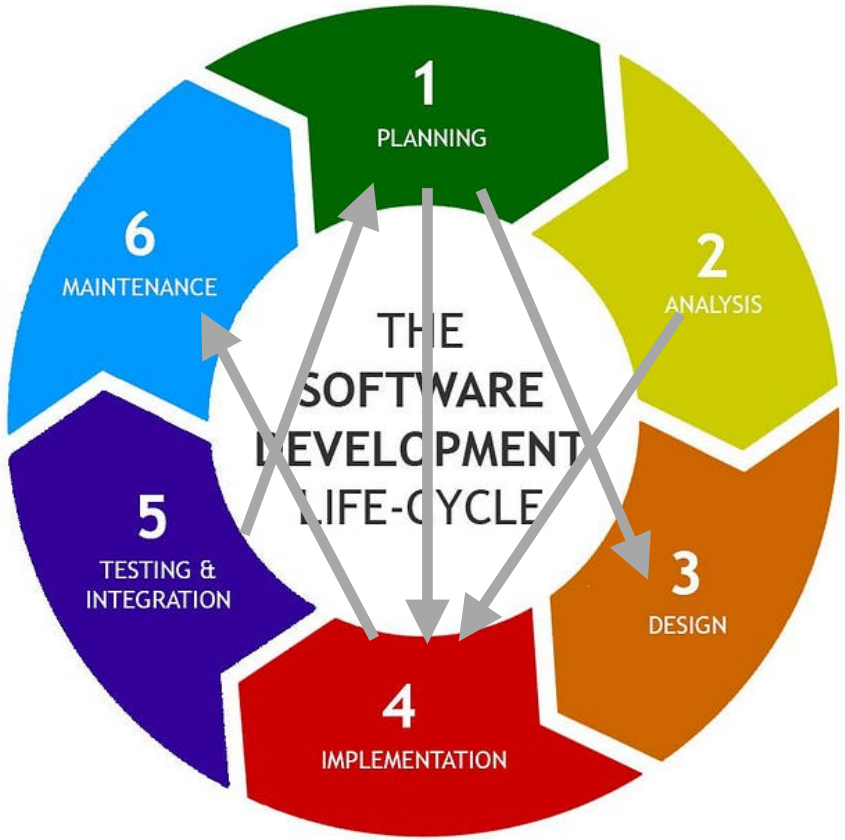
Advantages

- Faster time-to-market
- Testing and superior quality
- Flexible priorities
- Lower risk
- Project visibility and transparency, etc.,

SDLC (Software Development Life Cycle)

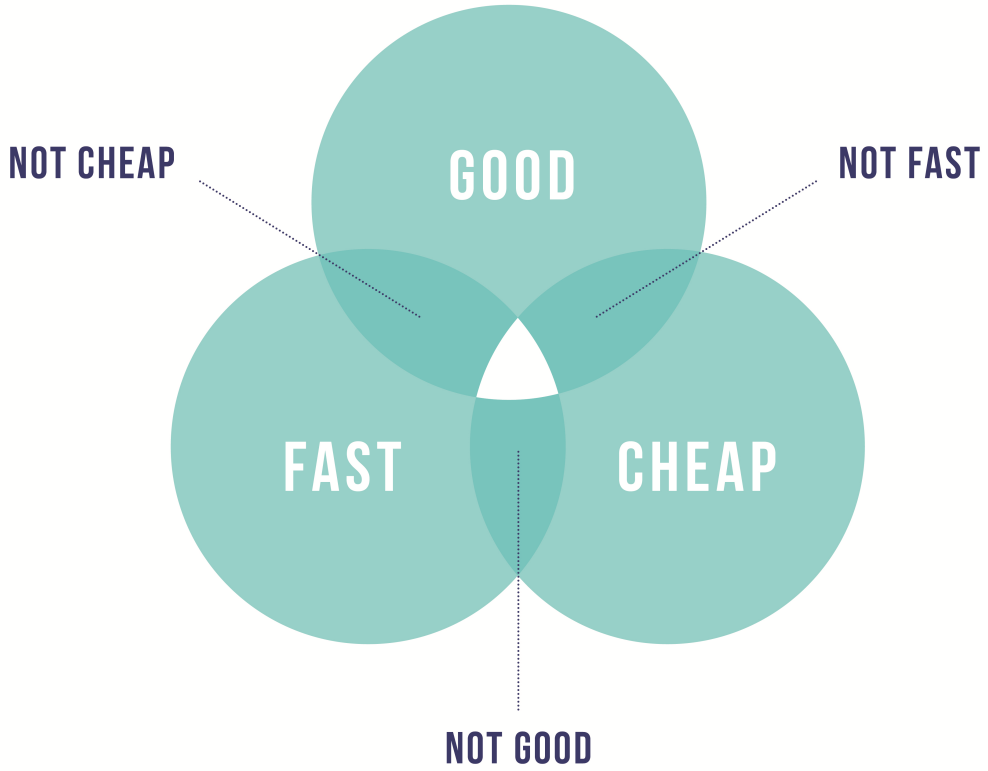


SDLC (Software Development Life Cycle)

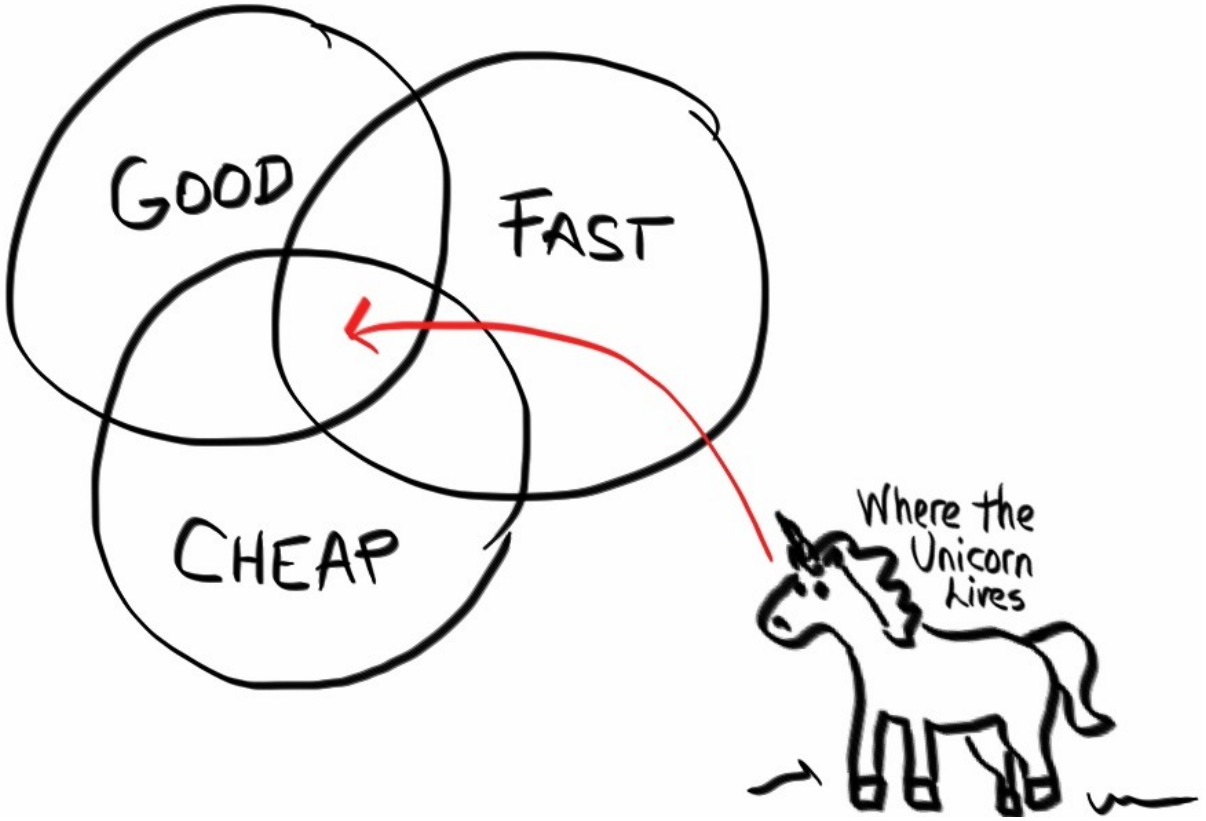


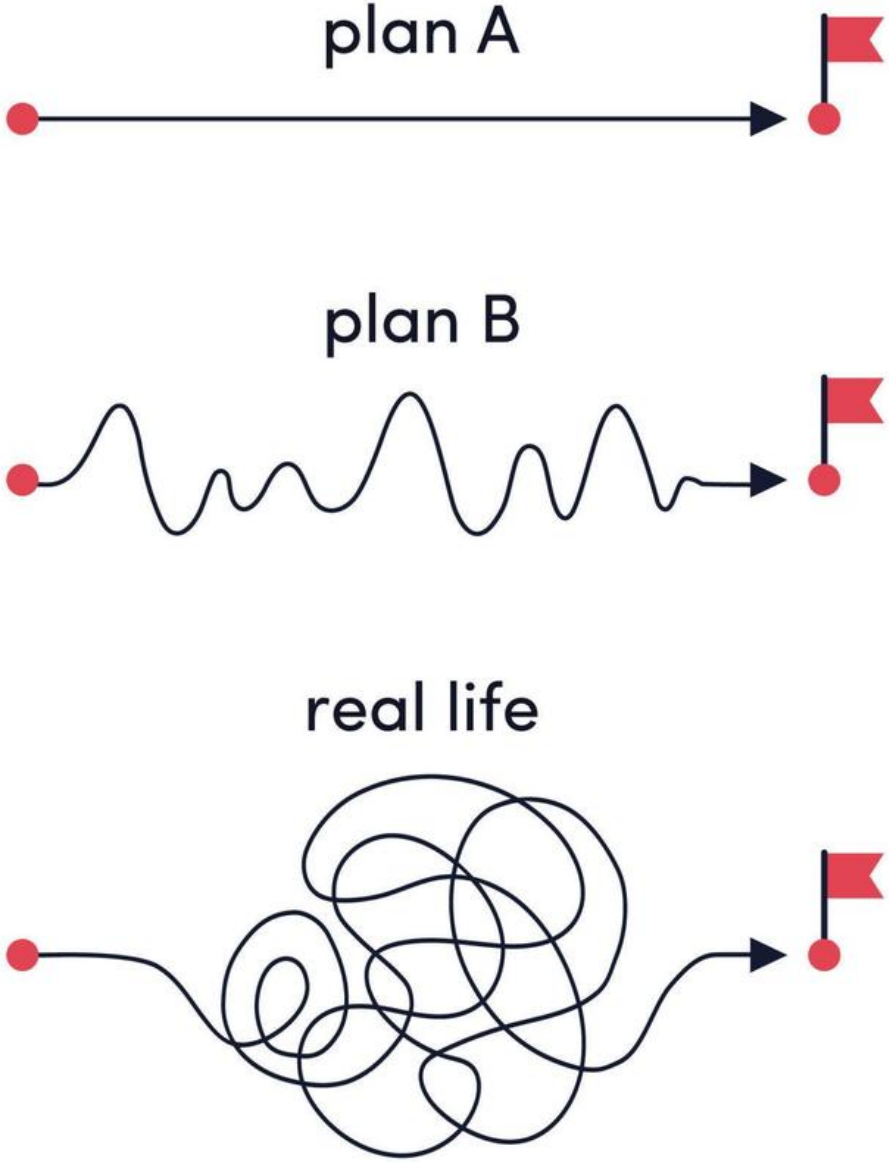
Cost vs Time vs Quality

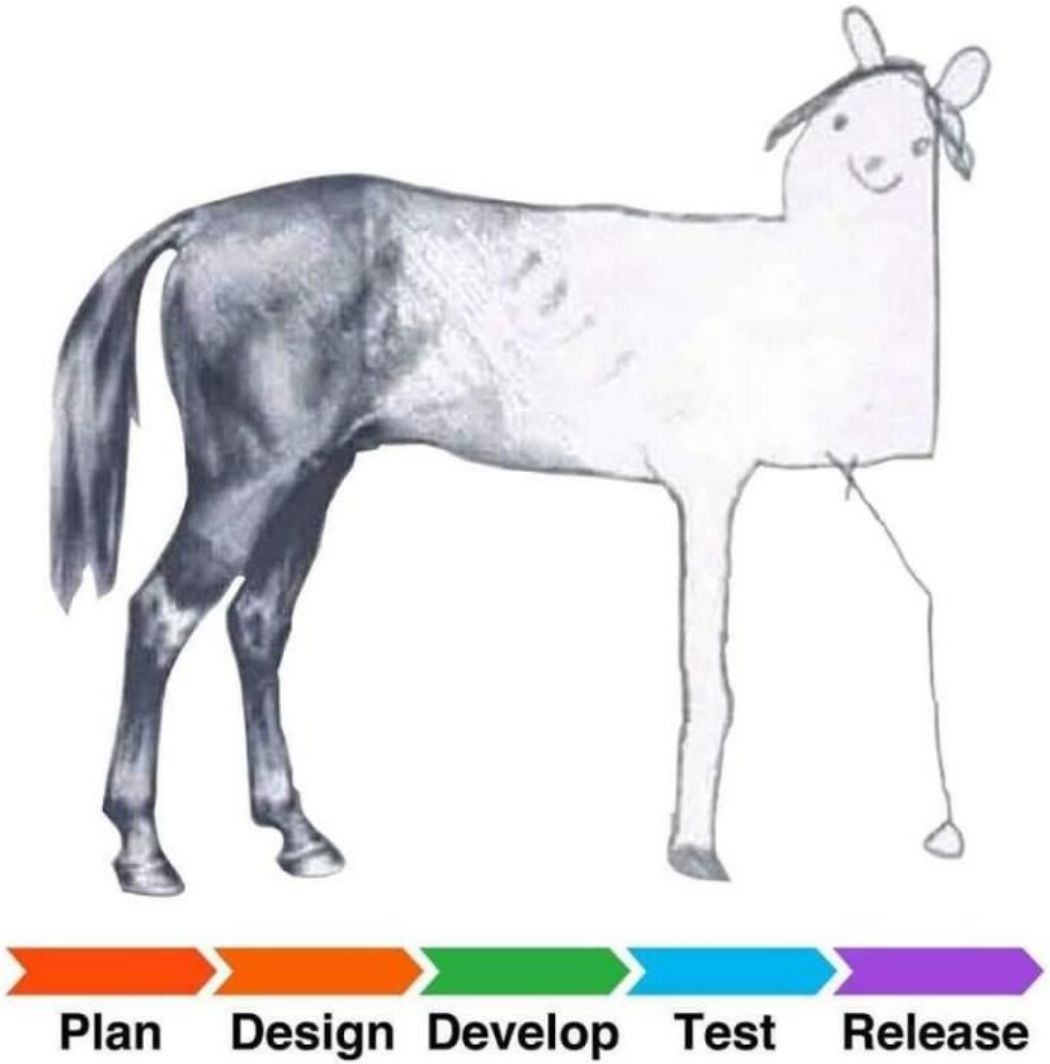
Cost vs Time vs Quality



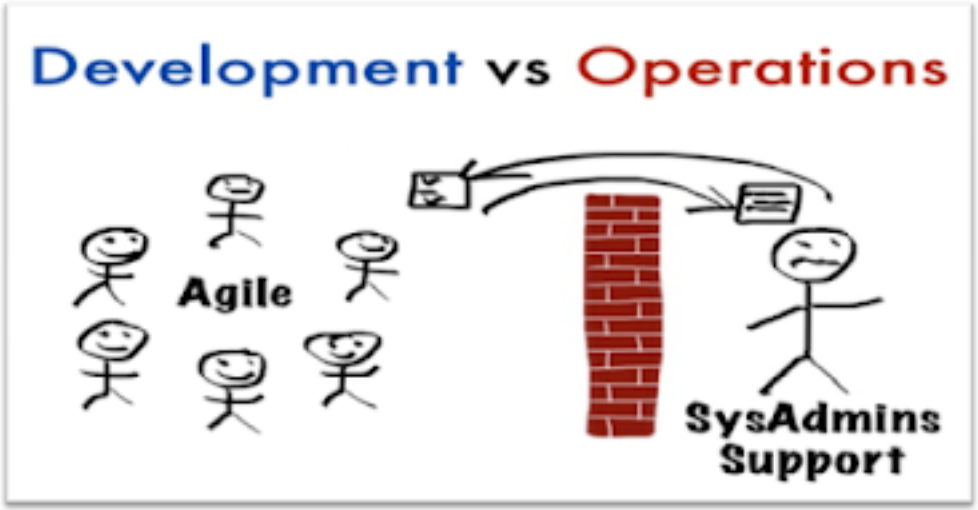
Cost vs Time vs Quality







Development vs Operations



Development vs Operations

3 C’s of Software Quality

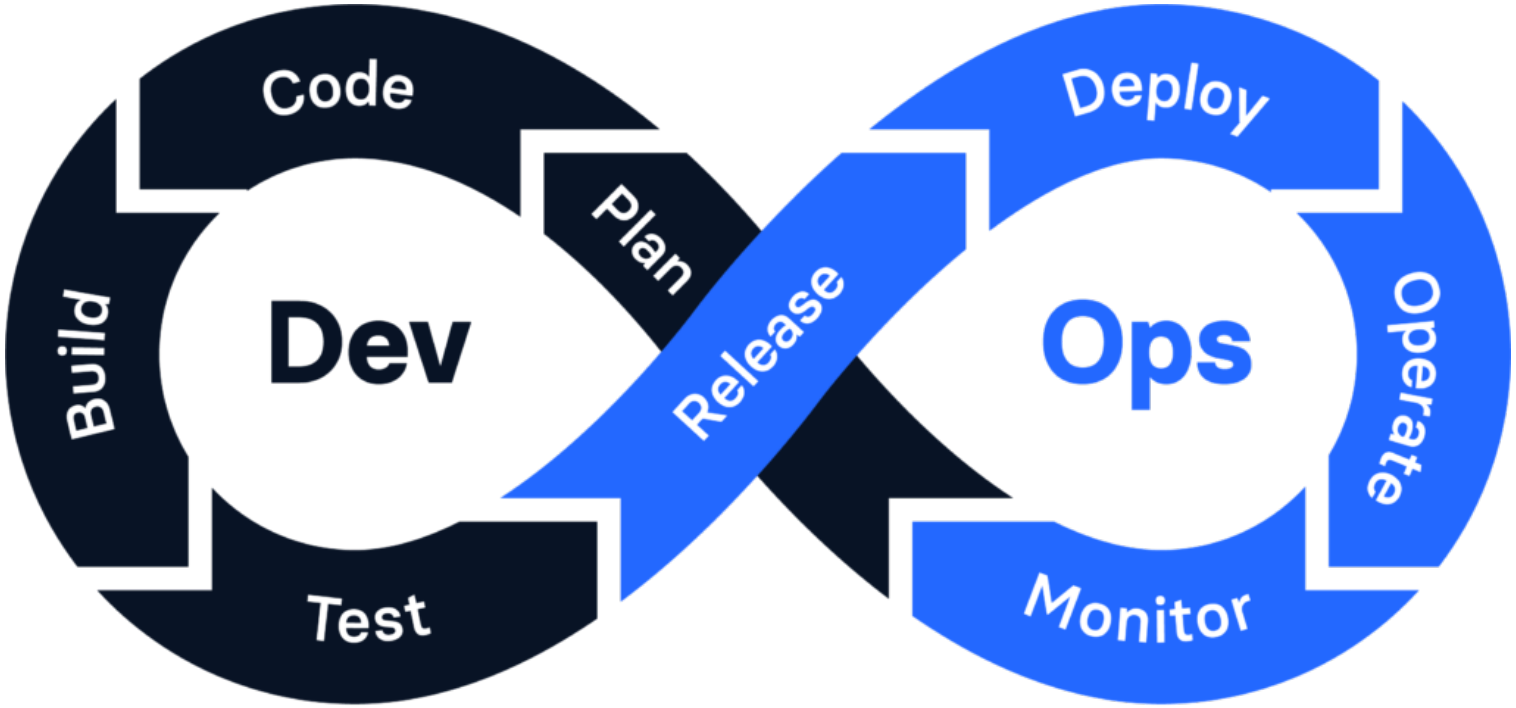
- **Correctness**: perform its intended functions correctly and without errors or bugs
- **Completeness**: fulfills all its specified requirements. Ensures meets the needs of users
- **Consistency**: Ensures that the software is reliable and can be used in different situations.

Problems

- Cannot deliver on time
- Deliver software with low quality
- Low customer satisfaction
- Collaboration in team have problem
- Automate not common and/or cannot do repeat

What’s DevOps ?

DevOps



What's DevOps

The Microsoft-approved definition of DevOps

- " the **union of people**, processes and technology to **continually provide value** to customers. "

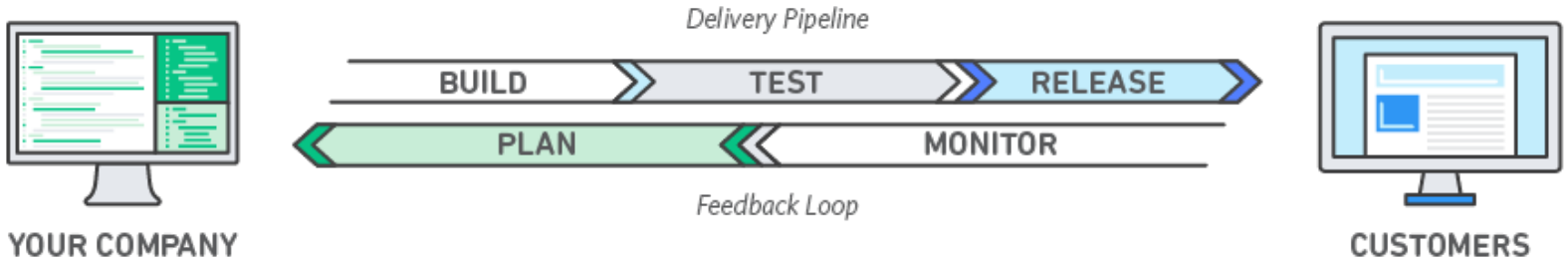
What's DevOps

Patrick Debois (The Father of DevOps)

- " It's a movement of **people** who think it's changing in the IT Industry time to stop wasting money, time to start delivering **great software**, and building systems that **scale** and last **DEV** Integration **Ops** Communication Collaboration. "

What's DevOps

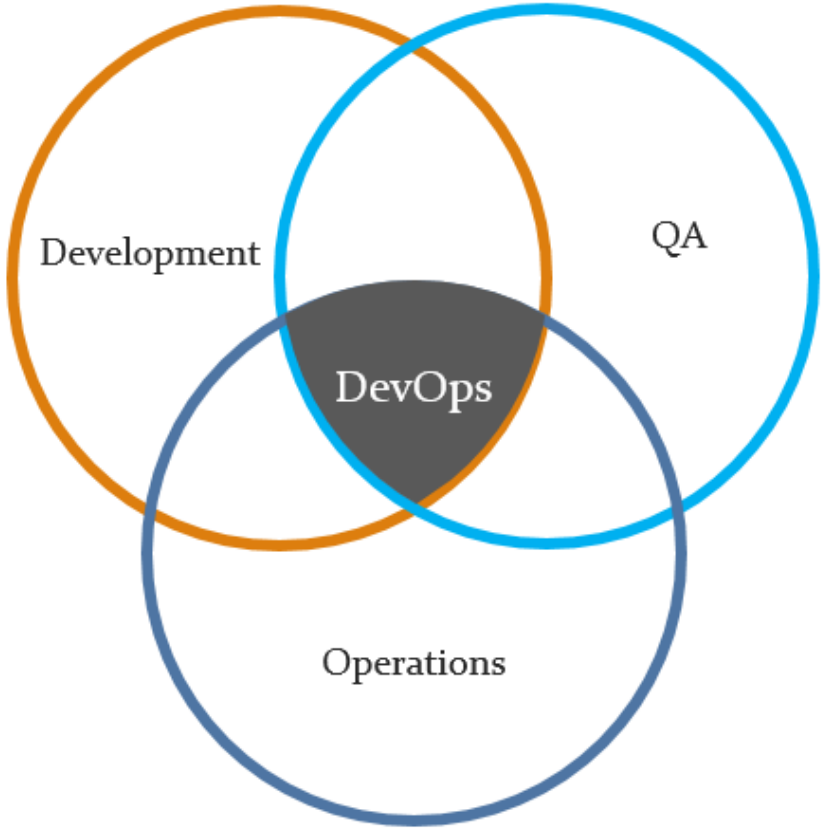
- DevOps emphasizes **people** (and culture), and seeks to improve collaboration between operations and development teams. DevOps implementations utilize technology— especially **automation** tools for **more rapid** and **more quality**



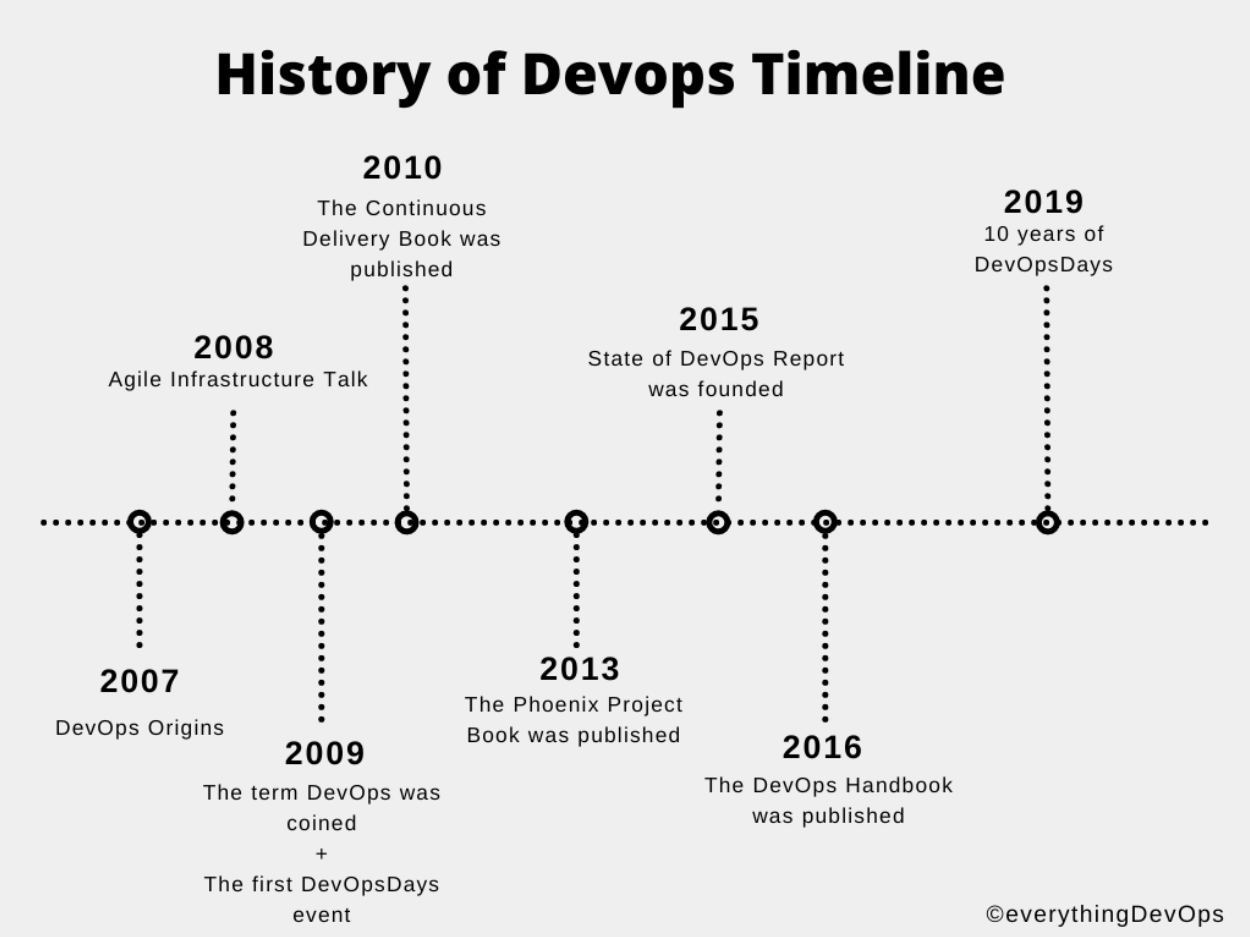
DevOps

- It’s not a product
- It’s not a job title
- It’s not all about tools

DevOps



History of DevOps



Benefits

- Faster Time-to-Market
- Improved Collaboration
- Increased Quality, Reliability
- Iteration & Continuous Feedback
- Better Customer Satisfaction

How to start DevOps ?

CALMS framework

CALMS framework

- Culture
- Automation
- Lean
- Measurement
- Sharing

Culture

- Open **communication** with inter and Internal team
- **Collaboration**: shared ownership and responsibility
- **Trust**: Teams and individuals
- **Empathy**: Understanding the challenges faced

Automation

- Continuous Integration (CI)
- Continuous Delivery (CD)
- Infrastructure as Code (IaC)

Lean

- Small Batches
- Continuous Improvement
- Eliminating repeated tasks

Measurement

- Lead time for changes
- Deployment frequencies
- Meantime to restore
- Failure rate

DORA Metrics

Software delivery performance metric	Elite	High	Medium	Low
Deployment frequency For the primary application or service you work on, how often does your organization deploy code to production or release it to end users?	On-demand (multiple deploys per day)	Between once per week and once per month	Between once per month and once every 6 months	Fewer than once per six months
Lead time for changes For the primary application or service you work on, what is your lead time for changes (i.e., how long does it take to go from code committed to code successfully running in production)?	Less than one hour	Between one day and one week	Between one month and six months	More than six months
Time to restore service For the primary application or service you work on, how long does it generally take to restore service when a service incident or a defect that impacts users occurs (e.g., unplanned outage or service impairment)?	Less than one hour	Less than one day	Between one day and one week	More than six months
Change failure rate For the primary application or service you work on, what percentage of changes to production or released to users result in degraded service (e.g., lead to service impairment or service outage) and subsequently require remediation (e.g., require a hotfix, rollback, fix forward, patch)?	0%-15%	16%-30%	16%-30%	16%-30%

Sharing

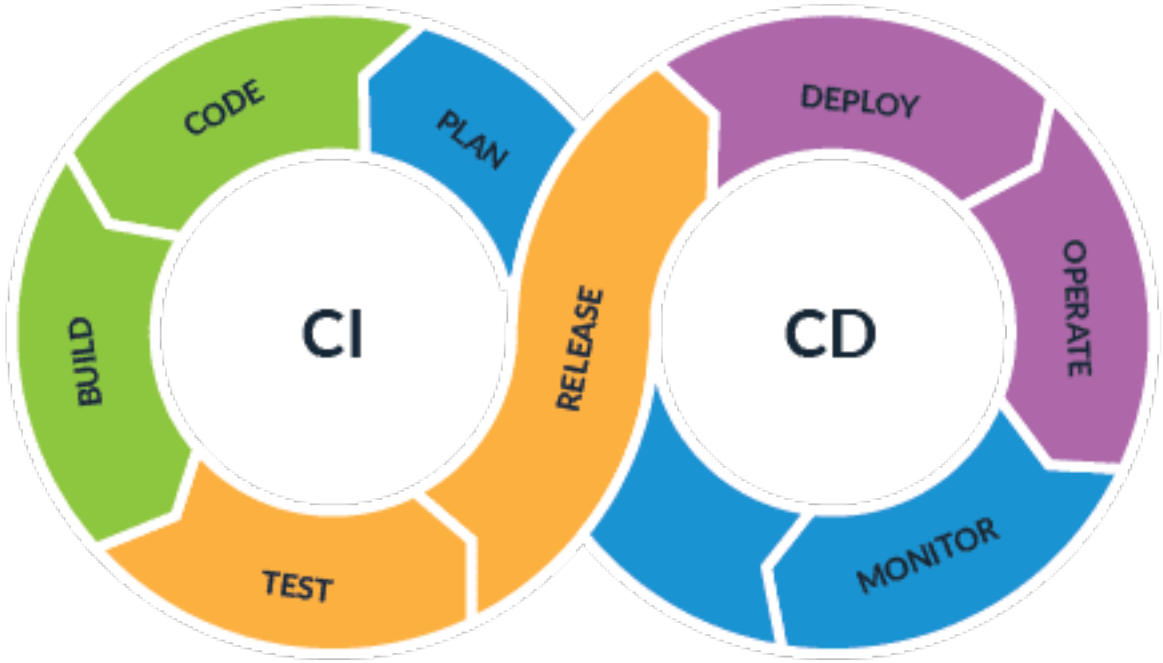
- Knowledge Exchange
- Documentation

CI/CD

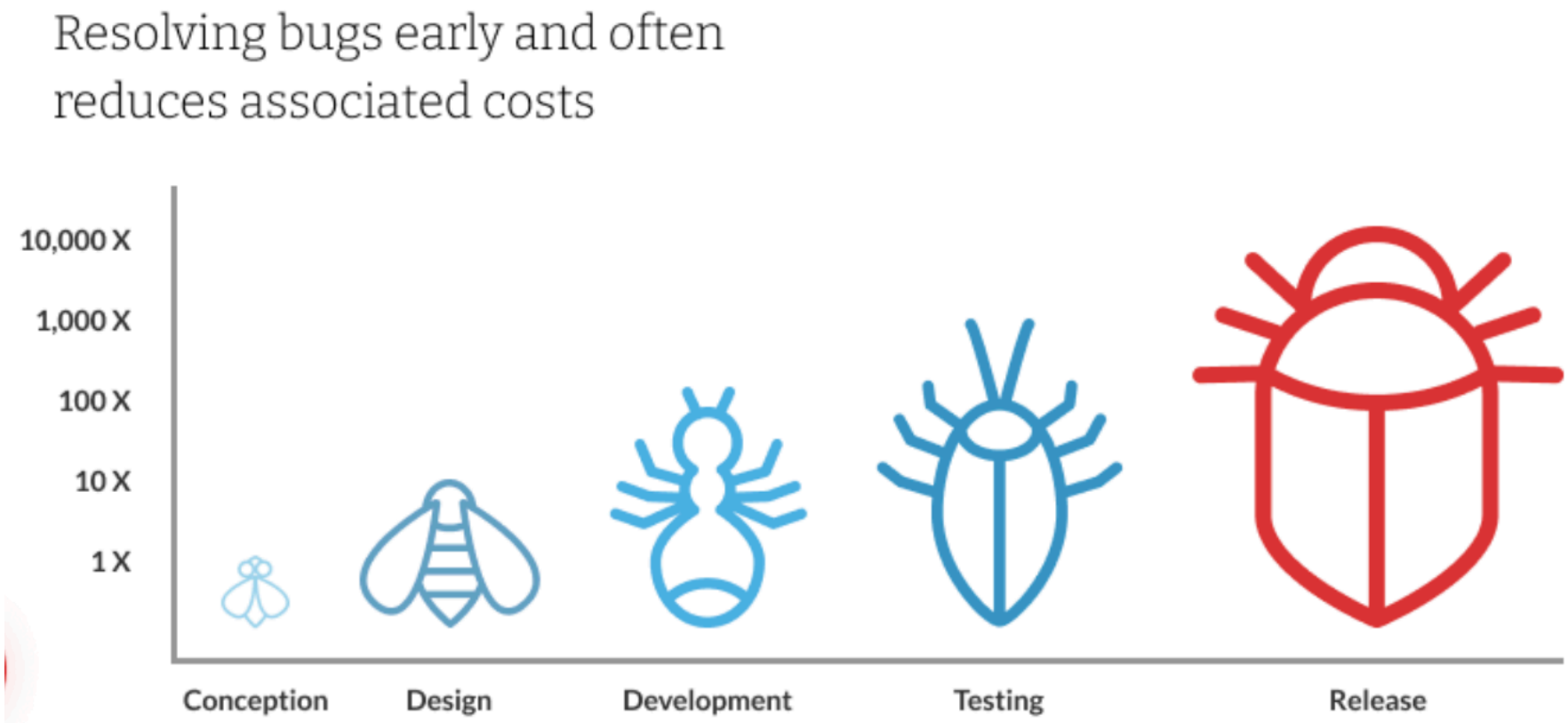
What is a CI/CD pipelines ?

- Continuous Integration and Continuous Deployment (CI/CD) pipeline represents an **automatic workflow** that continuously integrates code developed by software developers and **deploys** them into a target environment with less human intervention.

What is a CI/CD pipelines ?



Cost of integration

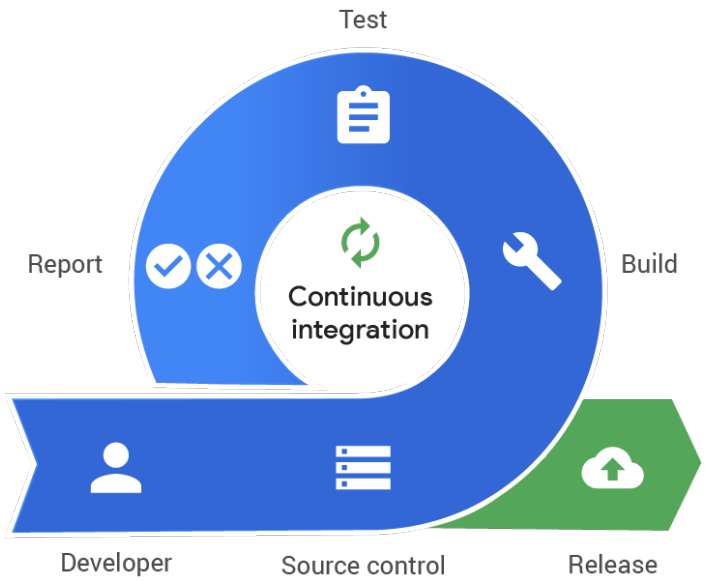


Continuous Integration (CI)

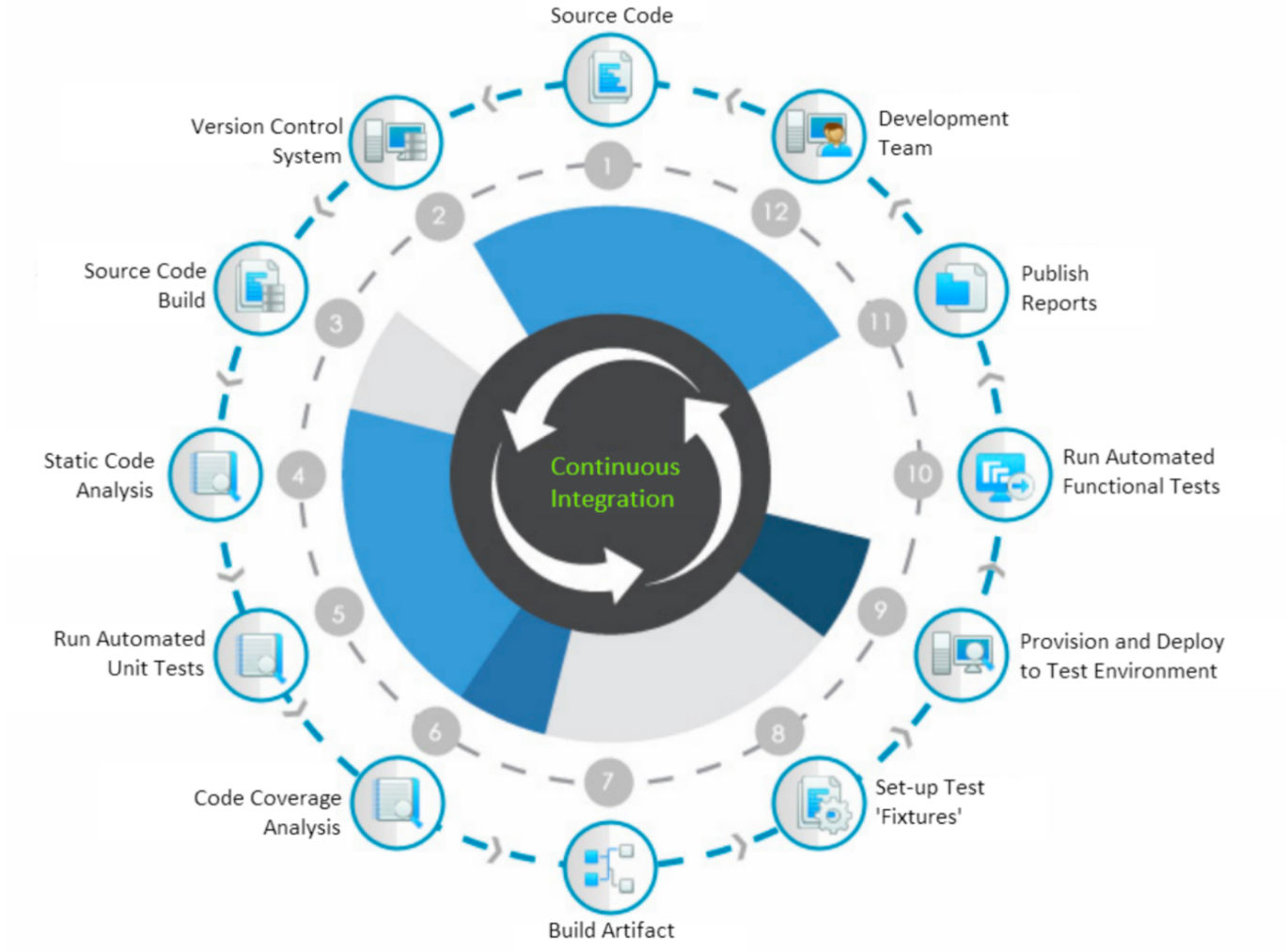
Continuous Integration (CI)

CI is part of the development process. **Automatically** build the software whenever new software is **committed** by developers

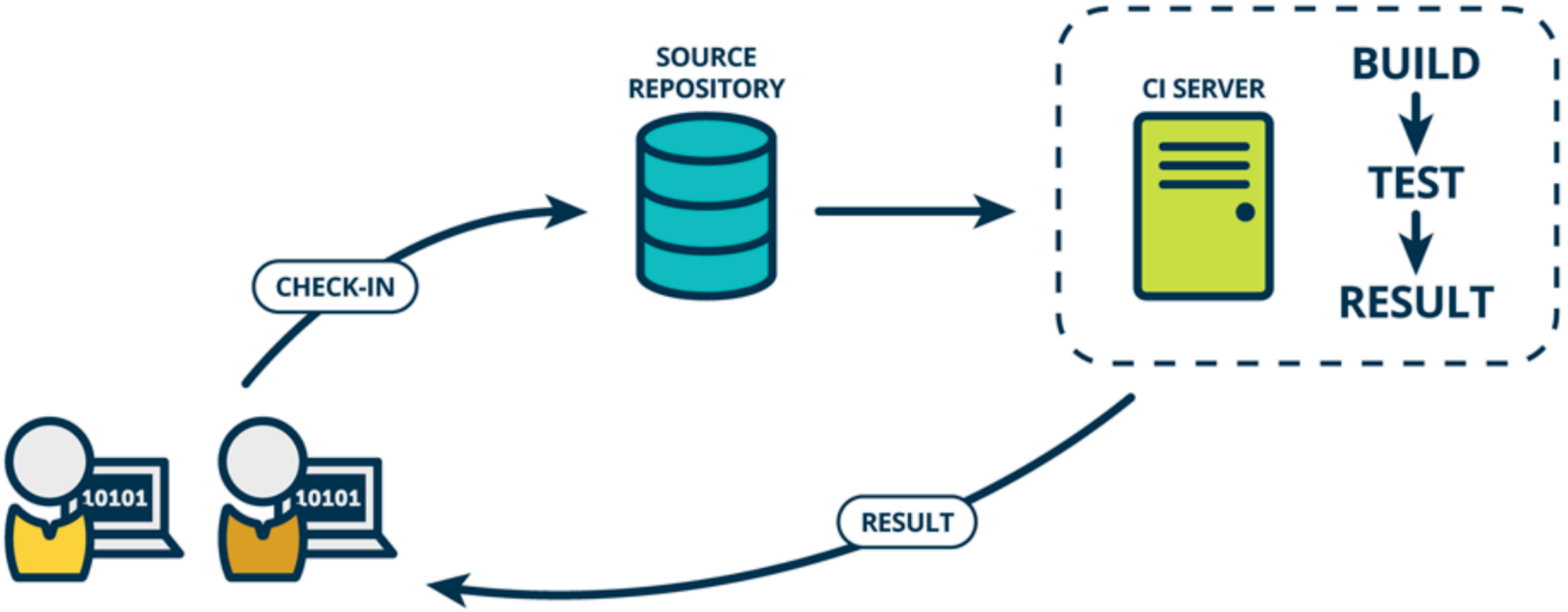
- Focuses on **frequently** merging code changes into a central repository
- Each code change, **trigger automated** builds and tests
- **Early** detection
- This ensures that the code compiles correctly and **passes all** the essential **tests**



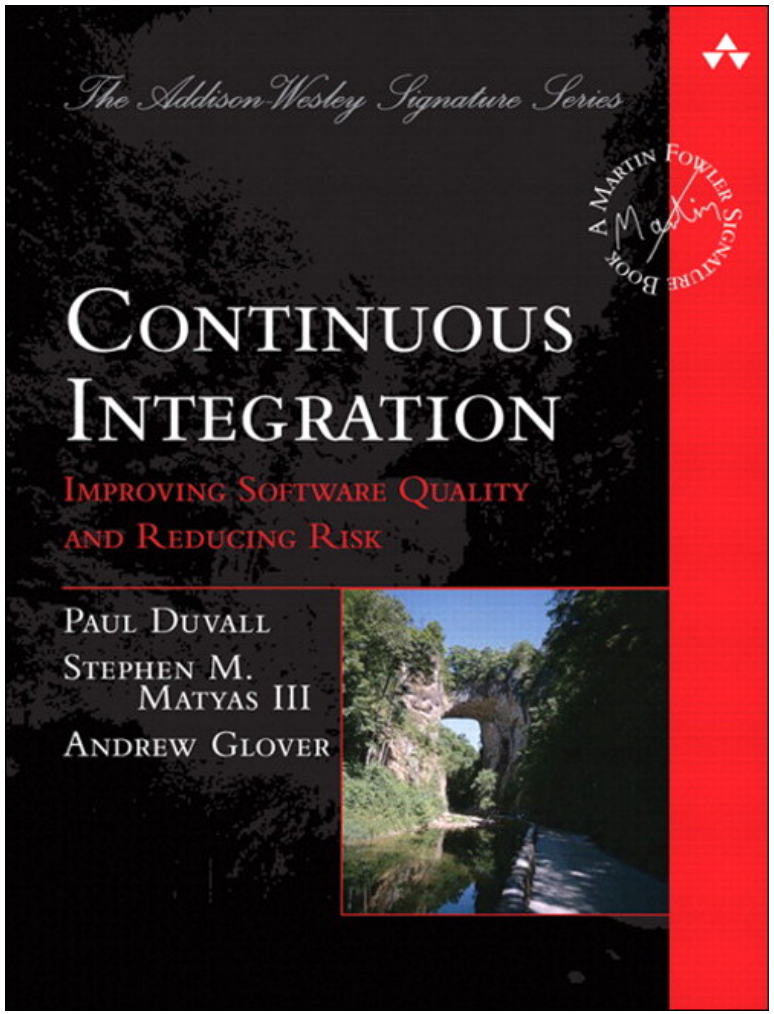
Continuous Integration (CI)



Continuous Integration (CI)



CI's practices



CI's practices

1. Maintain **a single** source repository

The key take away here is to have all artifacts needed to build the project in one repository. Not require additional dependencies or manual steps

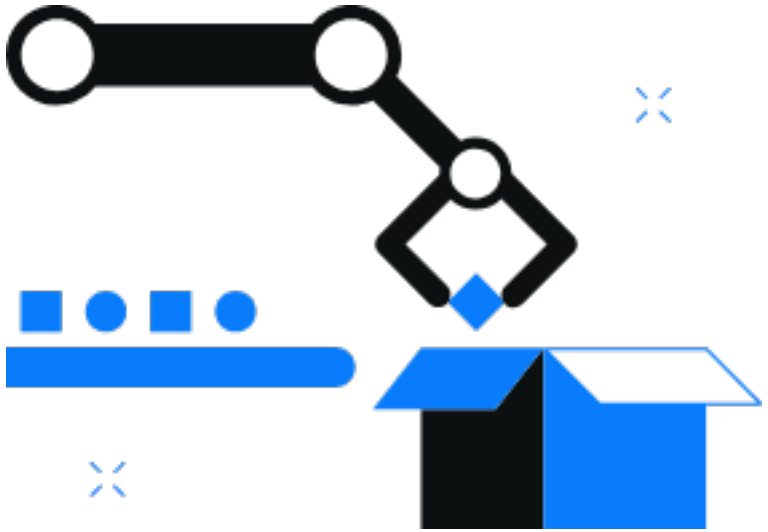
Caution

Recommends that **only a single branch** of the project, **NOT** to use **branching**. However depend on your organize with team

CI's practices

2. Automate the build

Automation of the build should include steps such as compiling the code, executing unit tests and integration tests



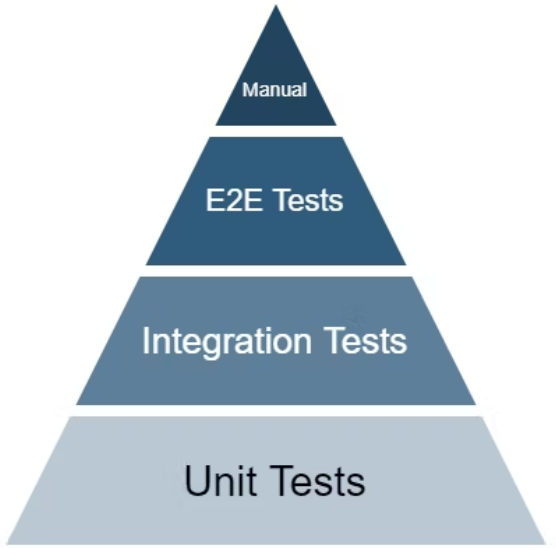
CI's practices

3. Make the build self-testing

All tests should run to confirm that it behaves as the developers expect it to behave, and always test on your machine first

Caution

Unit tests should test behavior, not implementation details



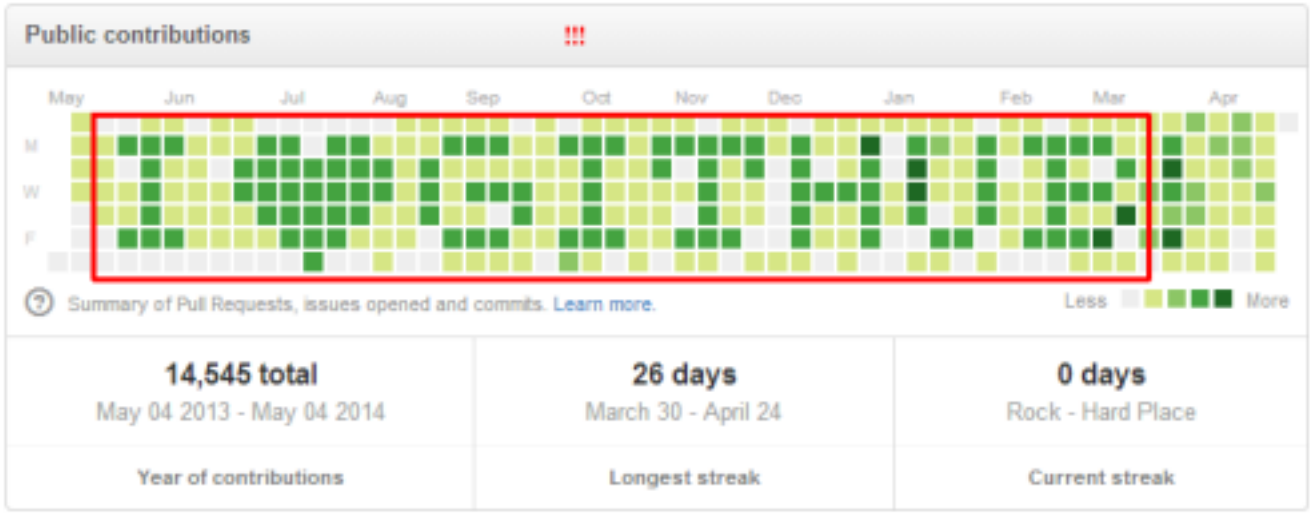
CI's practices

4. Everyone commits to the baseline every day, (frequency and small)

Every committer can reduce the number of conflicting changes.

Caution

Do not commits source code that do not work



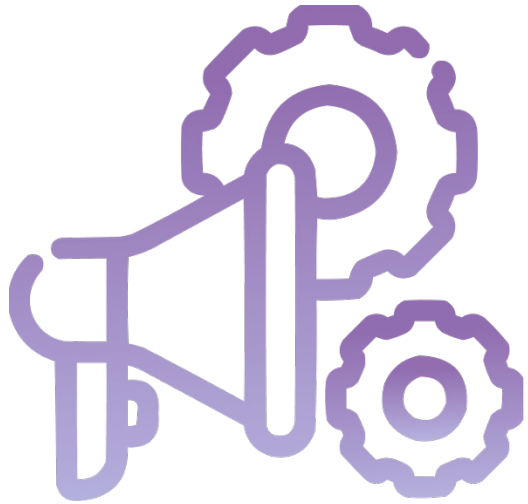
CI's practices

5. Every commit (to baseline) should trigger a build

You should separate the CI workflows. This includes the steps from compilation right up to packaging and testing.

Caution

There should be no commented out tests in the mainline branch



CI's practices

6. Fix Broken Builds Immediately

Nobody has a higher priority task than fixing the build

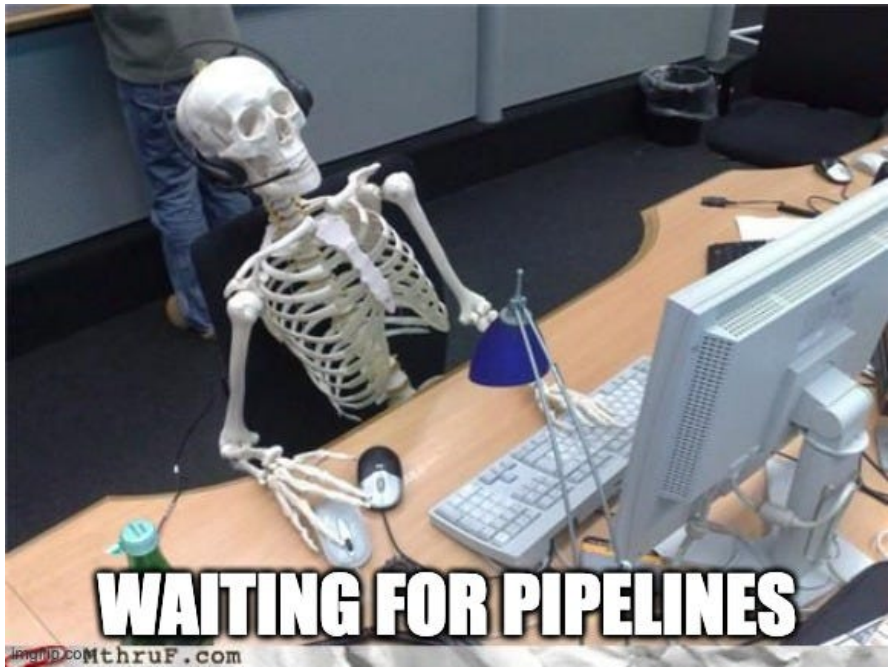
CI's practices

7. Keep the build fast

You should aim to have faster-executing tests than slow tests. This would mean that you need to have more unit tests than other types of tests

Caution

Balancing your workflows with build fast



CI's practices

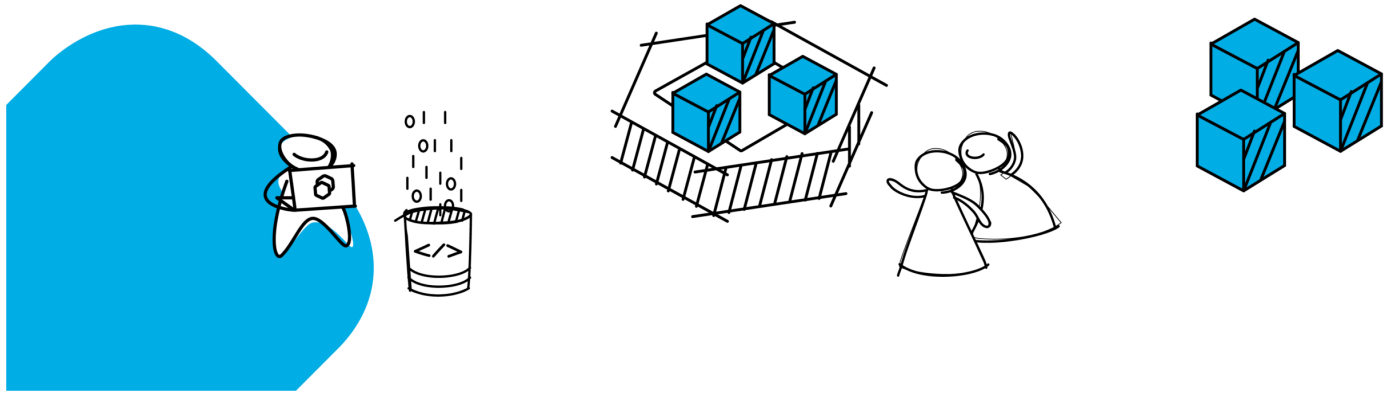
8. Test in a clone of the production environment

This is the hardest principle to put into practice in real world development. This needs the build automation system to create and deploy the packages into a staging environment that reflects the real production environment

CI's practices

9. Make it easy to get the latest deliverables (artifacts)

Make sure well known place where people can find



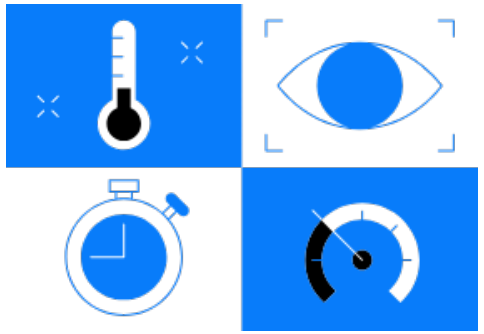
CI's practices

10. Everyone can see the results of the latest build

It should be easy to find out whether the build breaks and, if so, who made the relevant change. recommended that the emails are sent to the whole team when the build fails so that it can be fixed as soon as possible.

Caution

CI is not only for developers. The various metrics that can be derived using to improve quality of CI not just the quality of the software.



CI's practices

11. Automate deployment

Most CI systems allow the running of scripts after a build finishes. In most situations, it is possible to write a script to deploy the application to a live test server that everyone can look at

CI’s benefis

- Early Bug Detection
- Improved Code Quality
- Reduced Integration Issues
- Automated Build and Testing
- Improve Developer Productivity
- Consistency in Environments

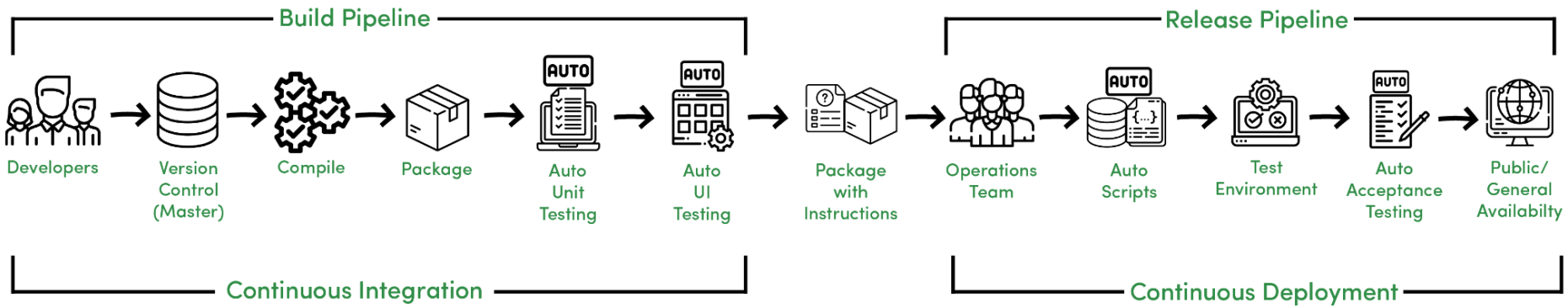
Workshop Continuous Integration

Continuous Deployment (CD)

Continuous Deployment (CD)

Continuous deployment stage refers to pulling the artifact from the repository and deploying it frequently and safely. Automate the deployment into the each environments with less human intervention

- Once the code passes CI checks, CD takes over
- Some approaches call Continuous Delivery: Makes the built code readily available for deployment, allowing for manual review before pushing it live

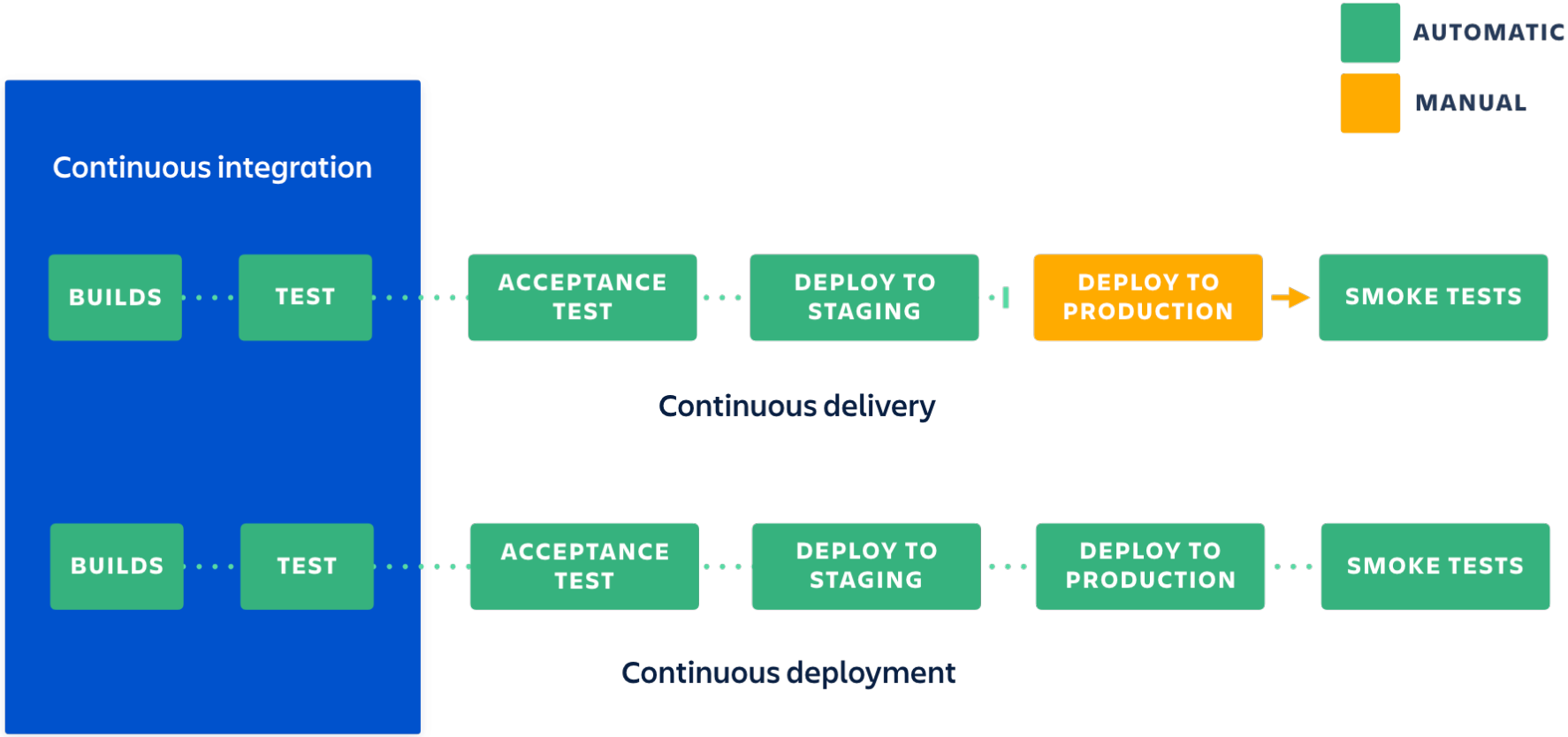


Continuous Delivery vs Continuous Deployment

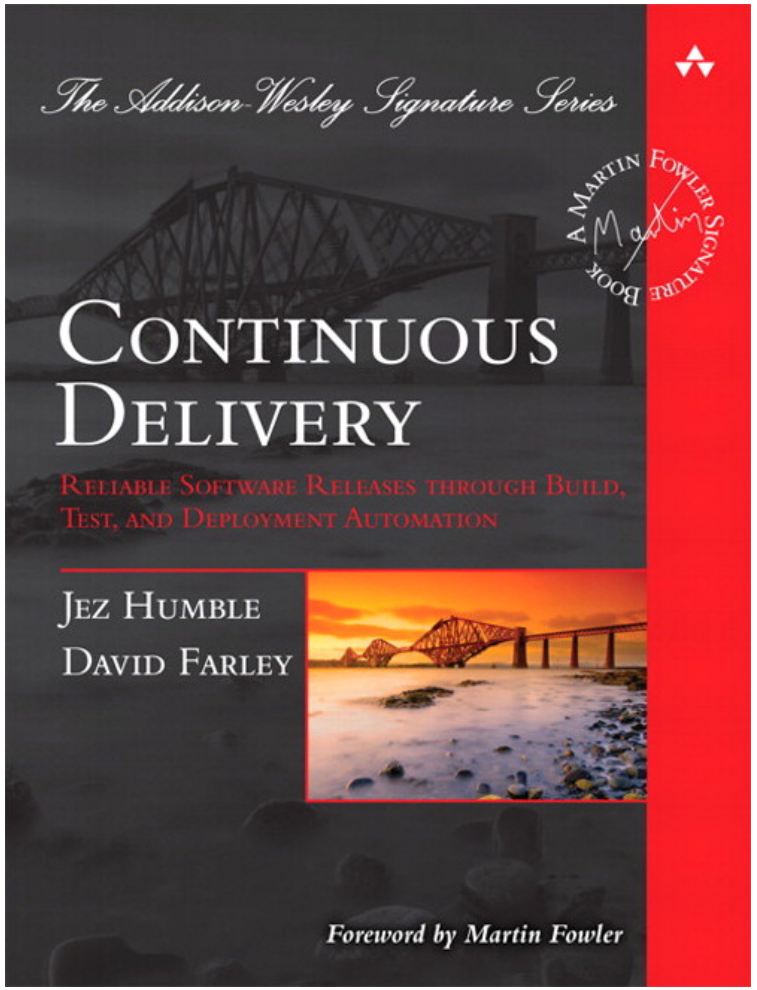
Both is an extension of continuous integration where code changes are automatically prepared for a release to production

But continuous delivery is the presence of a **manual approval** to update to production. However continuous deployment, production happens **automatically** without explicit approval

Continuous Delivery vs Continuous Deployment



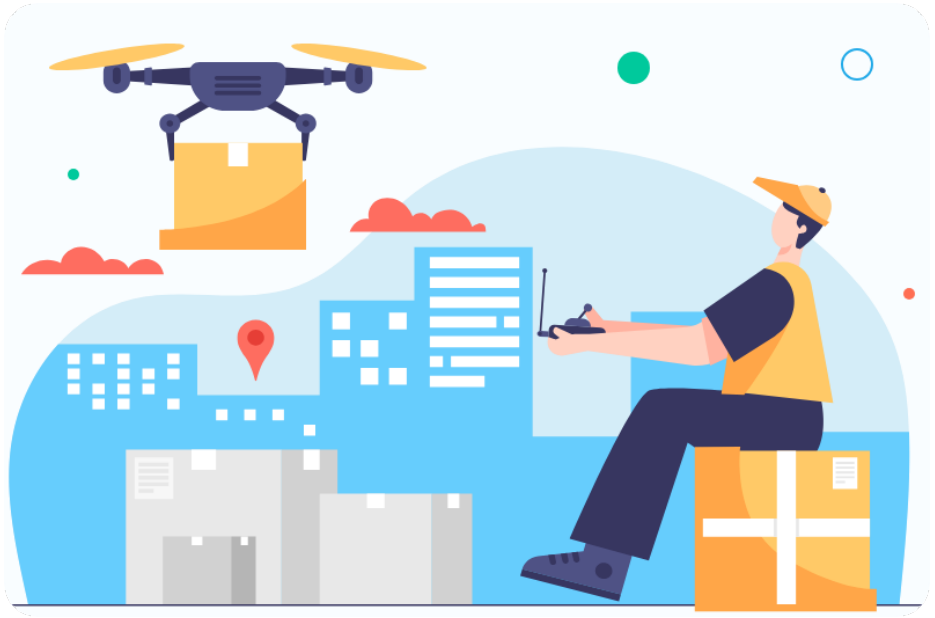
CD’s practices



CD’s practices

1. Automated Deployments

They emphasize that the deployment pipeline should be fully automated to ensure reliability and repeatability.



CD’s practices

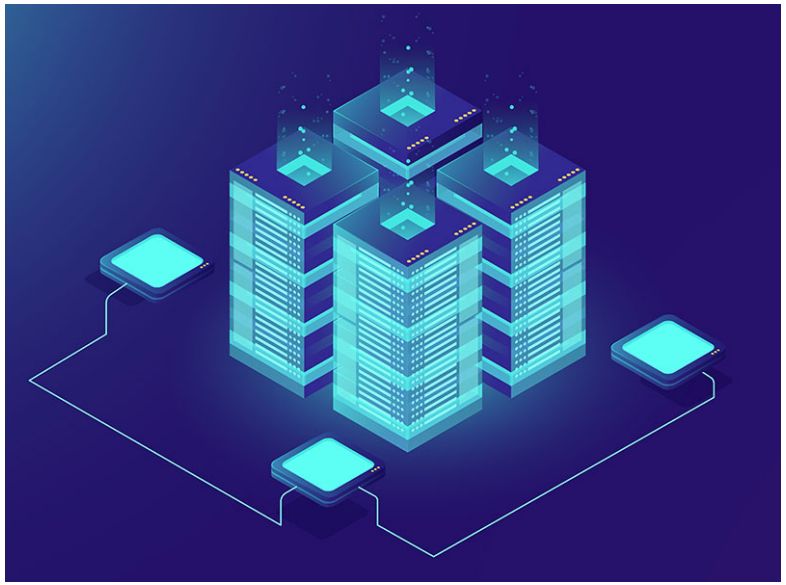
2. Version Control for All Artifacts

The importance of versioning all components, including infrastructure and configuration, as part of continuous delivery.

CD’s practices

3. Deploy to Production-Like Environments Early

Need to test in production-like environments to reduce surprises in the actual production release.



CD’s practices

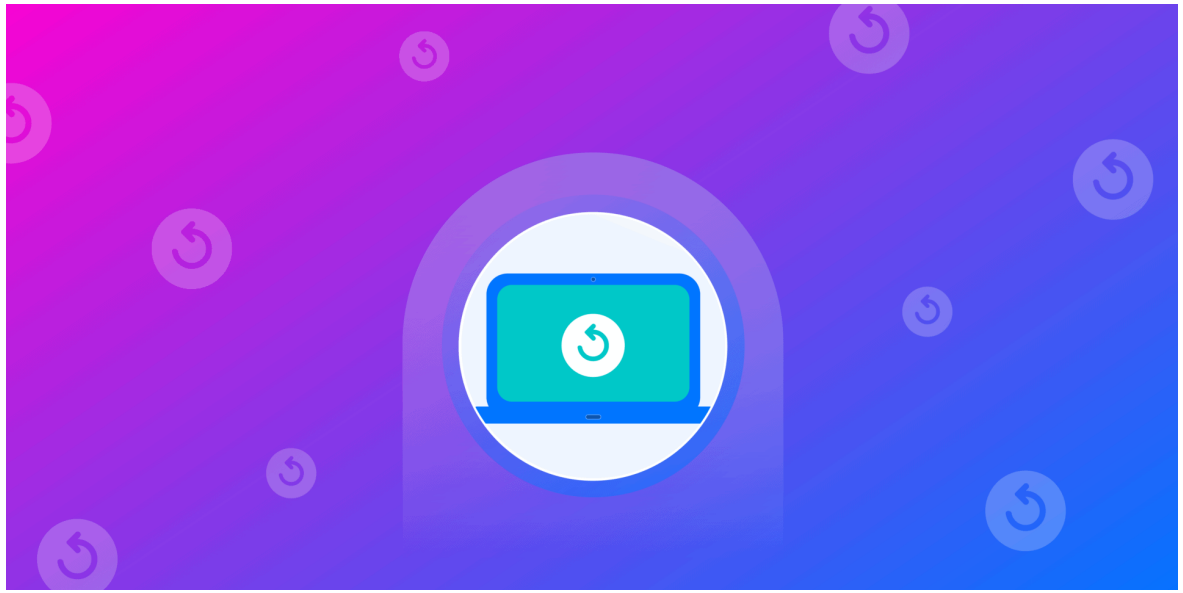
4. Zero-Downtime Deployments

Techniques like blue-green deployment and canary releases as methods to avoid downtime and reduce risk during deployments

CD’s practices

5. Rollback Mechanism

The need for a robust rollback mechanism to handle any issues arising in production



CD's benefits

- Automate the Software Release Process
- Reduced Manual Intervention, reduce human error
- Improve Developer Productivity
- Find and Address Bugs Quicker
- Deliver Updates Faster
- Easier Rollbacks

Deployment strategies

Deployment strategies

- Recreate deployment
- Ramped (Rolling deployment)
- Blue/green deployment
- Canary deployment
- A/B testing
- Shadow deployment

Recreate deployment

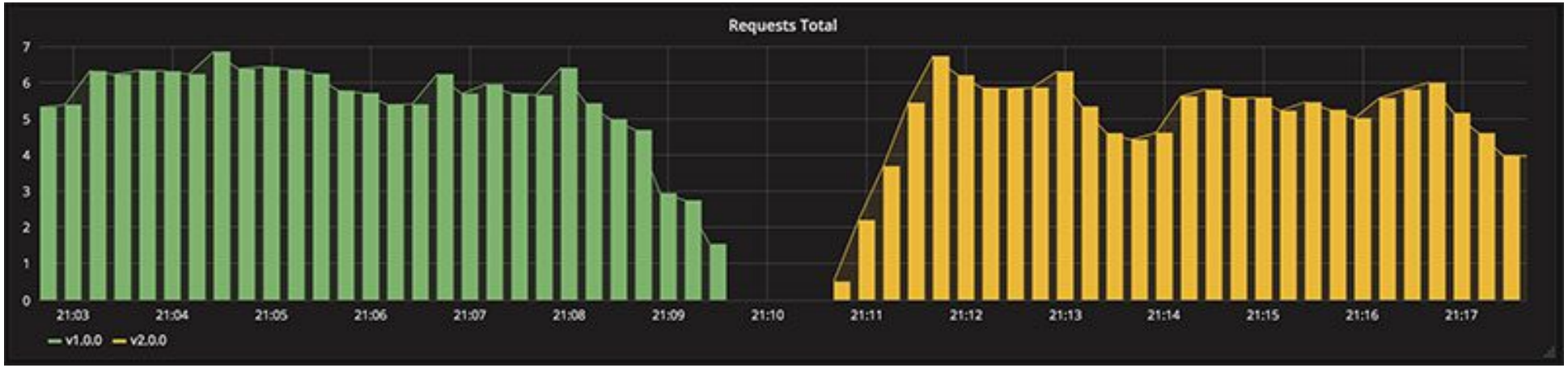
Terminating all the running instances then recreate them with the newer version

Advantage

- Application state entirely renewed

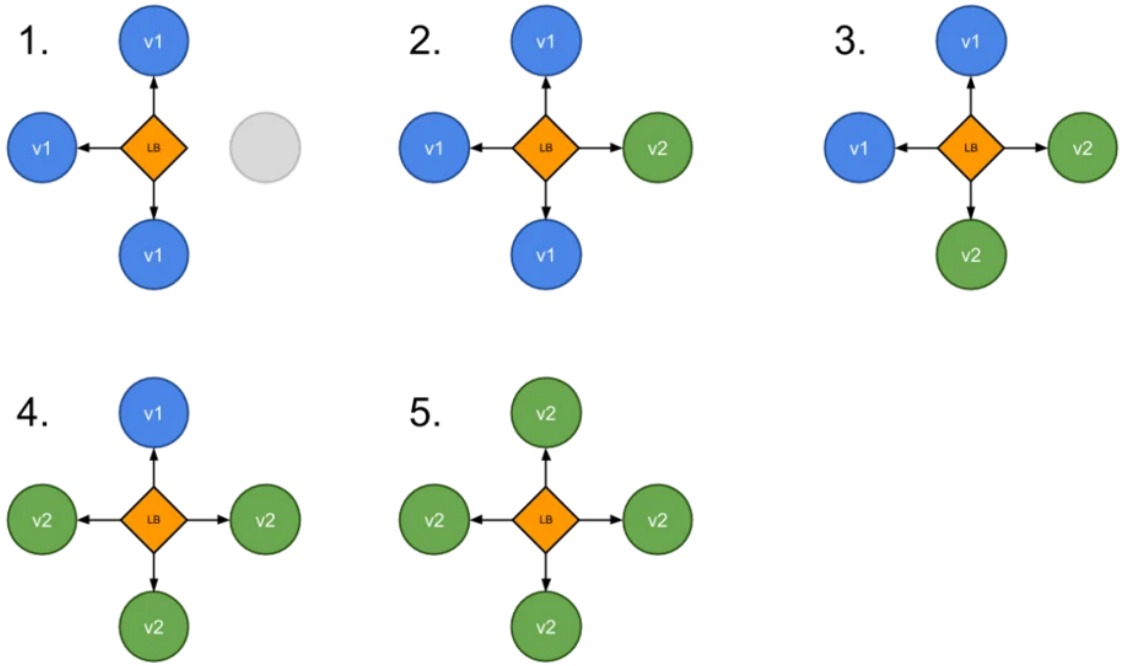
Disadvantage

- Downtime that depends on both shutdown and boot duration of the application



Ramped (Rolling deployment)

The rolling update is the **default** deployment strategy in Kubernetes that update the pods one by one without causing cluster downtime. the deployment objects create new pods using the new pod specifications and shut down the older ones when the application starts running successfully



Ramped (Rolling deployment)

Advantage

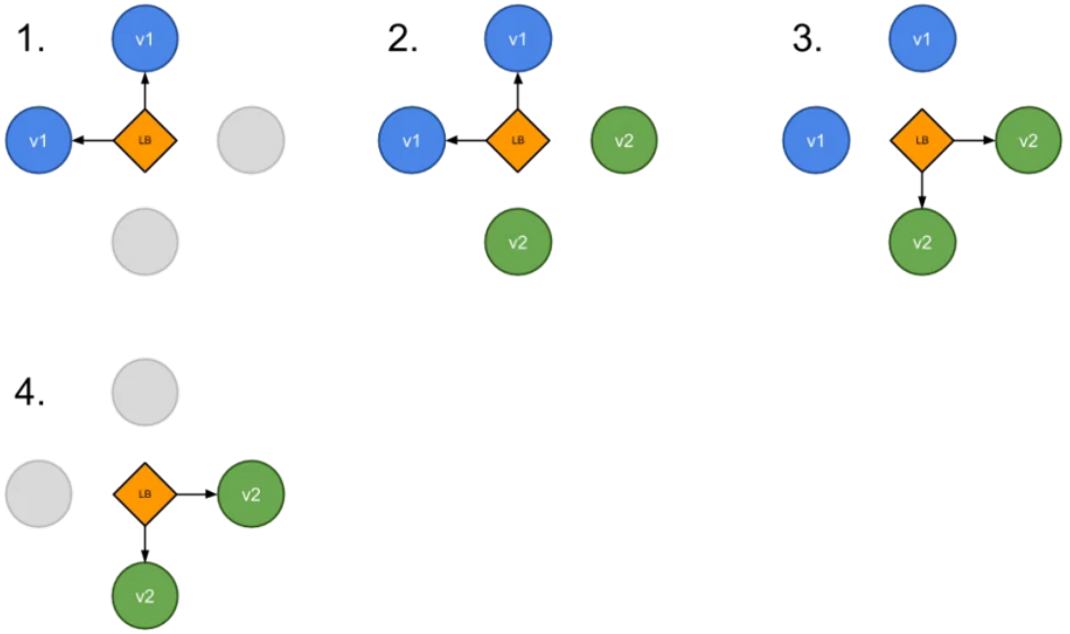
- Enables zero-downtime deployments
- Well-suited for large deployments and stateful applications

Disadvantage

- Rollout/rollback can take time
- Supporting multiple APIs is hard
- No control over traffic

Blue/green deployment

- While the “Blue” version is running, engineers deploy the “green” version in another deployment and run basic tests to ensure that the pods within the deployments are stable, then using services, you switch to the newer version of the application
- Best to avoid API versioning issues



Blue/green deployment

Advantage

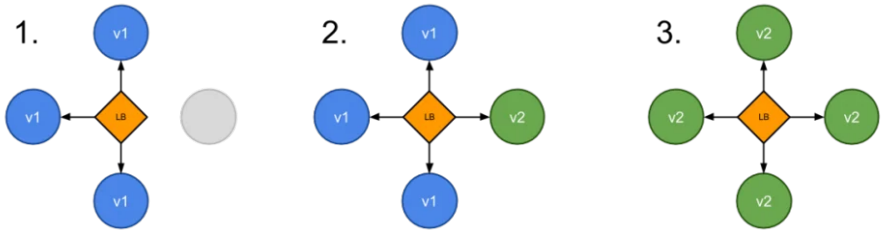
- Instant rollout/rollback
- Clear separation between old and new versions

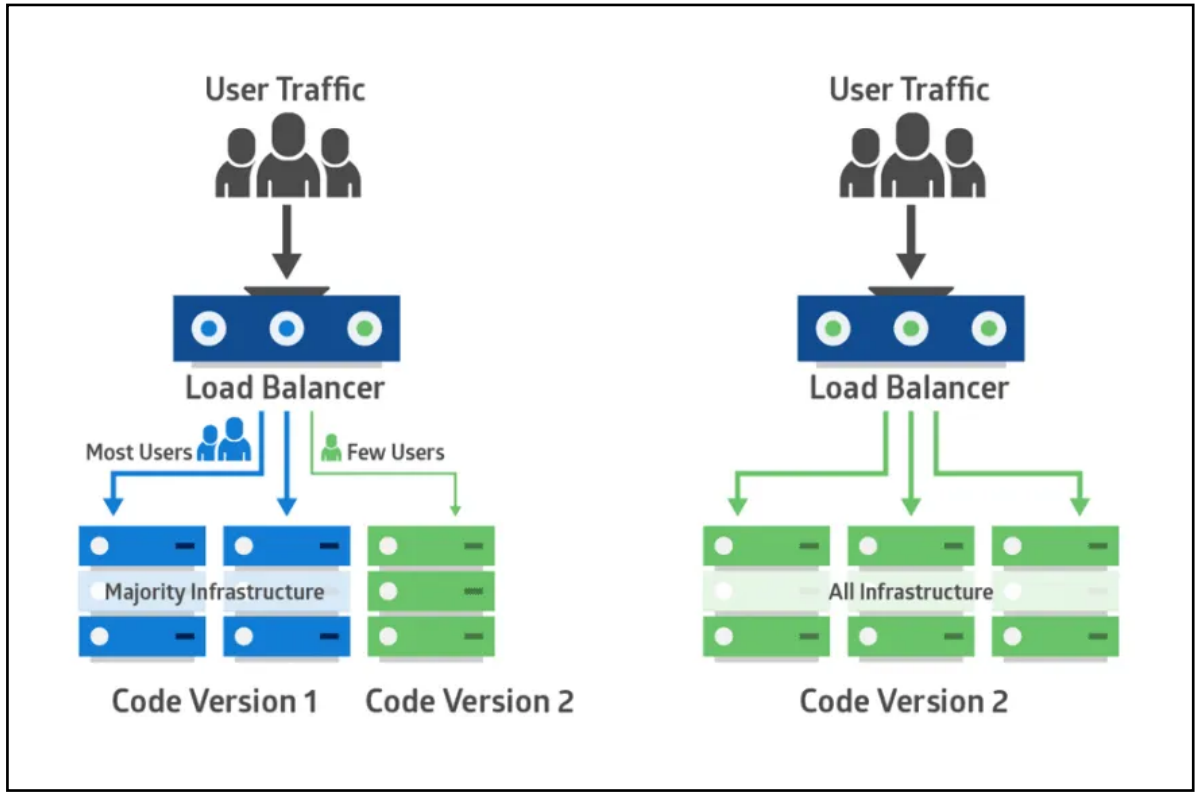
Disadvantage

- Requires double the resources
- More complex configuration

Canary deployment

- Similar to blue/green deployments, but only a small subset of traffic is directed to the new version (canary) initially. This allows for early detection of issues before a full rollout





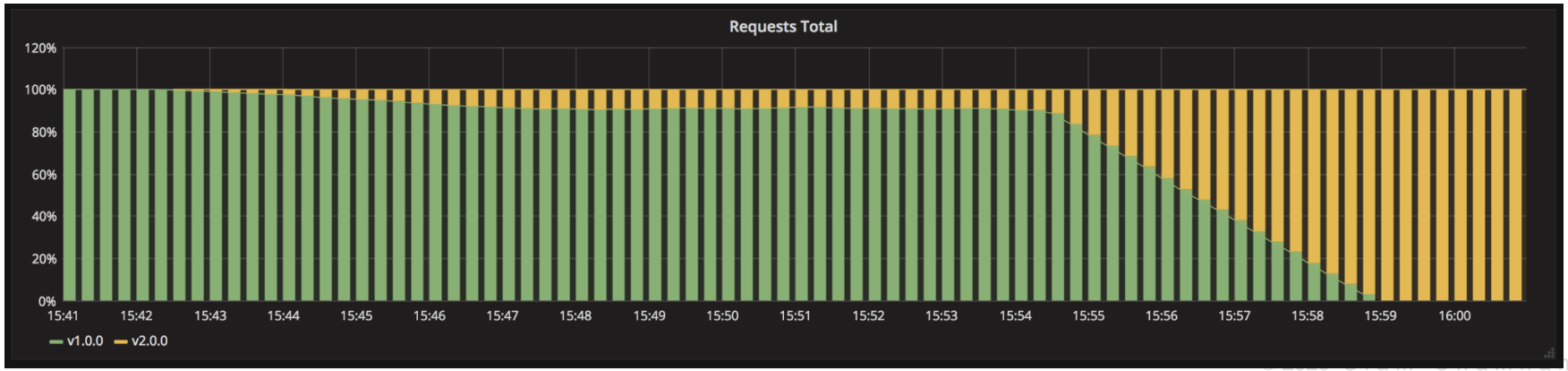
Canary deployment

Advantage

- Lowers risk by gradually rolling out new version to a small subset of users
- Enables early detection of issues

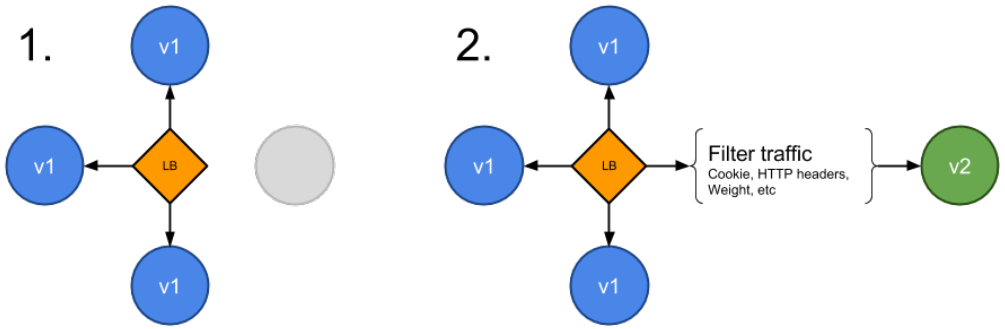
Disadvantage

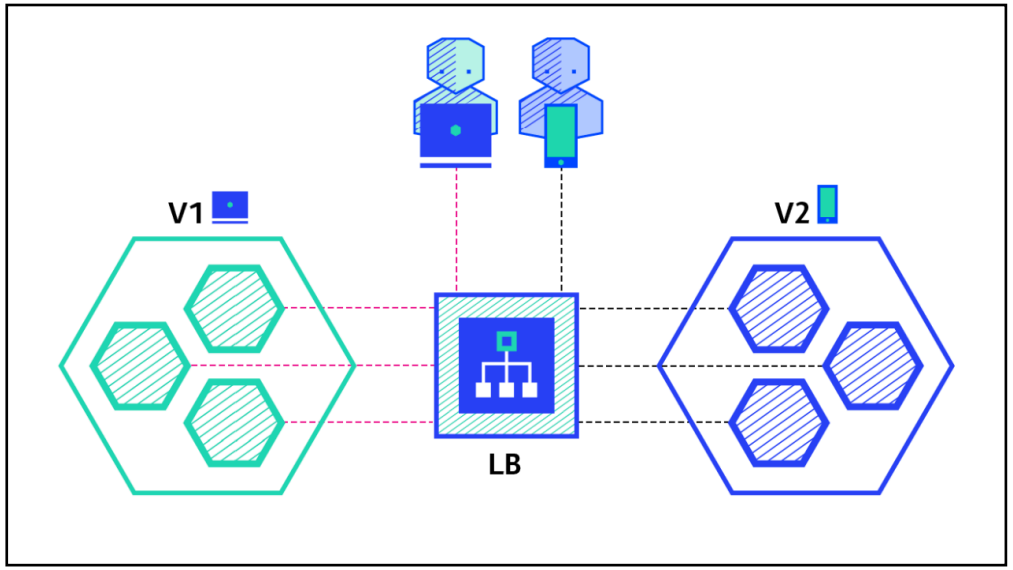
- Requires additional configuration for traffic routing
- Slow rollout, most of users cannot use new versions
- Application must support both of old & new versions



A/B testing

- A/B testing deployments consists of routing a subset of users to a new functionality under specific conditions. It is usually a technique for making business decisions based on statistics, rather than a deployment strategy.
- Best for feature testing on a subset of users





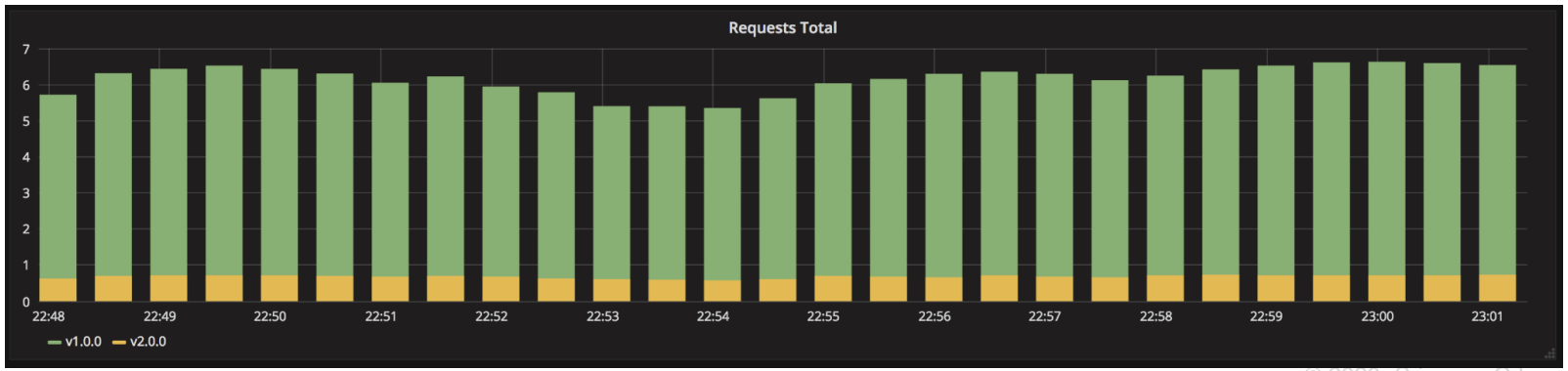
A/B testing

Advantage

- Several versions run in parallel
- Full control over the traffic distribution

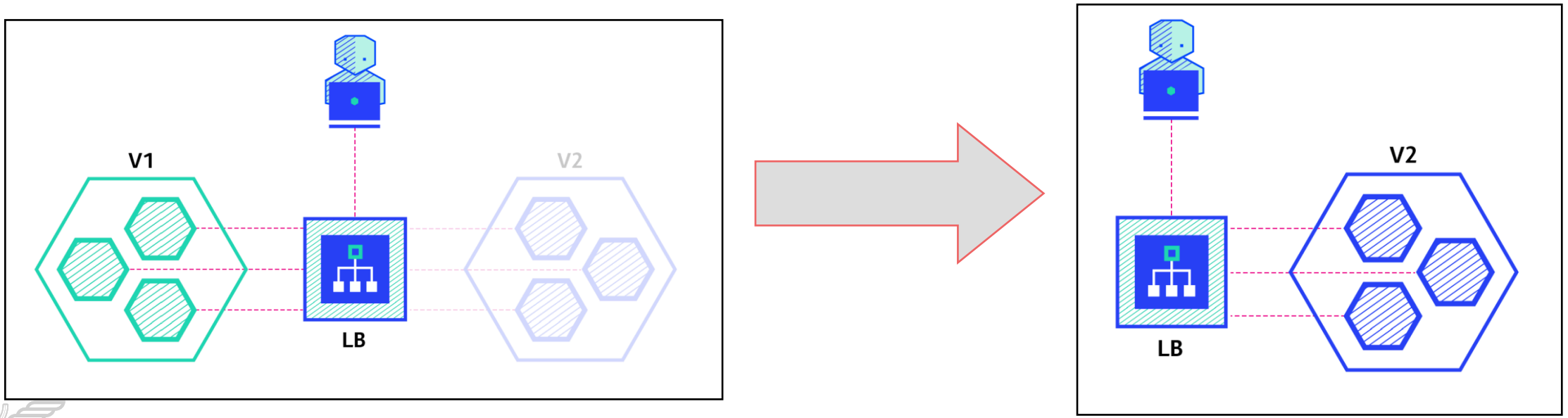
Disadvantage

- Requires intelligent load balancer
- Not straightforward, you need to setup additional tools
- Hard to troubleshoot errors for a given session



Shadow deployment

- Consists of releasing version B alongside version A, fork version A's incoming requests and send them to version B as well without impacting production traffic. This is particularly useful to test production load on a new feature



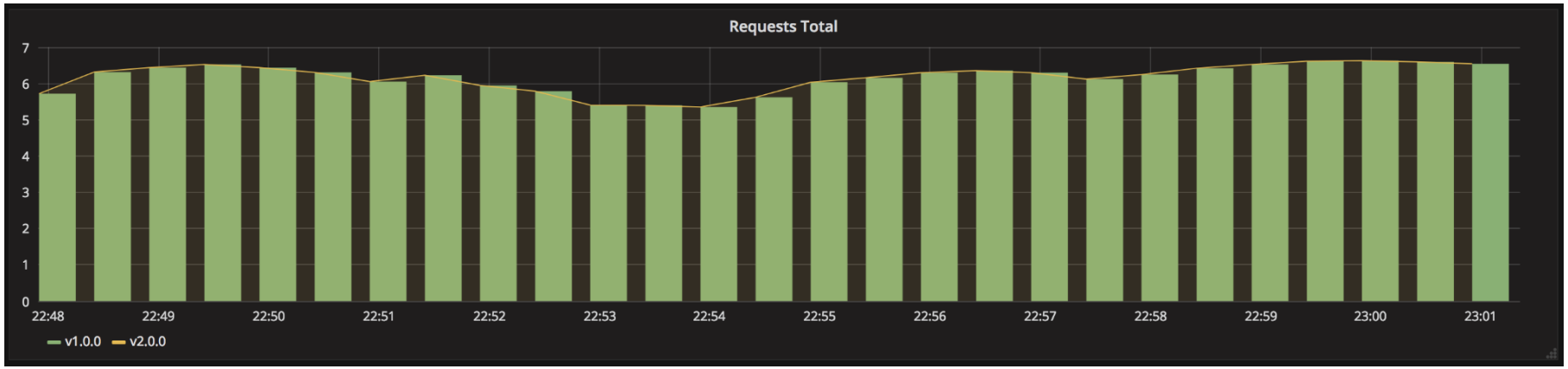
Shadow deployment

Advantage

- Performance testing, auto scale of the application with production traffic
- No impact on the user

Disadvantage

- Expensive as it requires double the resources.
- Complex to setup



DEPLOYMENT STRATEGIES

When it comes to production, a ramped or blue/green deployment is usually a good fit, but proper testing of the new platform is necessary.

Blue/green and shadow strategies have more impact on the budget as it requires double resource capacity. If the application lacks in tests or if there is little confidence about the impact/stability of the software, then a canary, a/b testing or shadow release can be used.

If your business requires testing of a new feature amongst a specific pool of users that can be filtered depending on some parameters like geolocation, language, operating system or browser features, then you may want to use the a/b testing technique.

Strategy	ZERO DOWNTIME	REAL TRAFFIC TESTING	TARGETED USERS	CLOUD COST	ROLLBACK DURATION	NEGATIVE IMPACT ON USER	COMPLEXITY OF SETUP
RECREATE version A is terminated then version B is rolled out	✗	✗	✗	■□□	■■■	■■■	□□□
RAMPED version B is slowly rolled out and replacing version A	✓	✗	✗	■□□	■■■	■□□	■□□
BLUE/GREEN version B is released alongside version A, then the traffic is switched to version B	✓	✗	✗	■■■	□□□	■■□	■■□
CANARY version B is released to a subset of users, then proceed to a full rollout	✓	✓	✗	■□□	■□□	■□□	■■□
A/B TESTING version B is released to a subset of users under specific condition	✓	✓	✓	■□□	■□□	■□□	■■■
SHADOW version B receives real world traffic alongside version A and doesn't impact the response	✓	✓	✗	■■■	□□□	□□□	■■■

Container Solutions

Workshop CI/CD with Jenkins

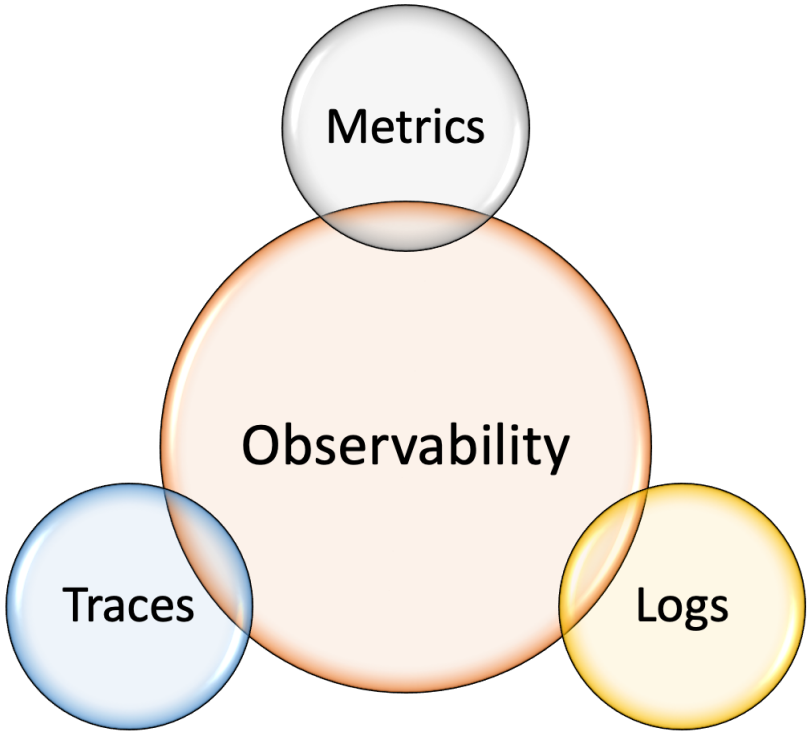
Challenges of implementing CI/CD

- Require organization changes and mindset shifts
- Deep technical knowledge to automate the entire process
- Team is required to use various technologies depend on application stack
- Time Investment: setting up an effective CI/CD pipeline can require significant initial time investment
- Frequent Failures: When adopting CI/CD, teams may initially experience frequent build and test failures due to process changes

Observability

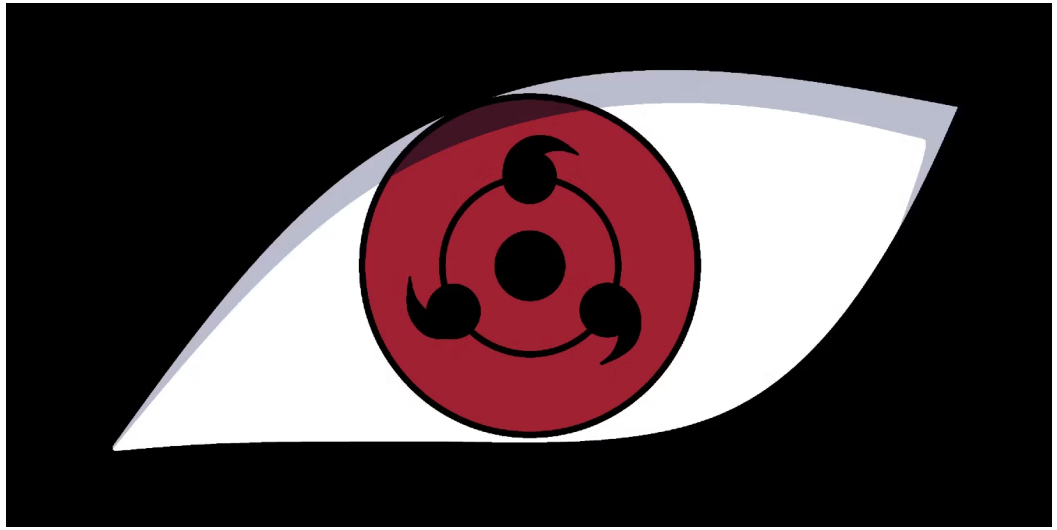
Three pillars of Observability

- Metrics
- Traces
- Logs

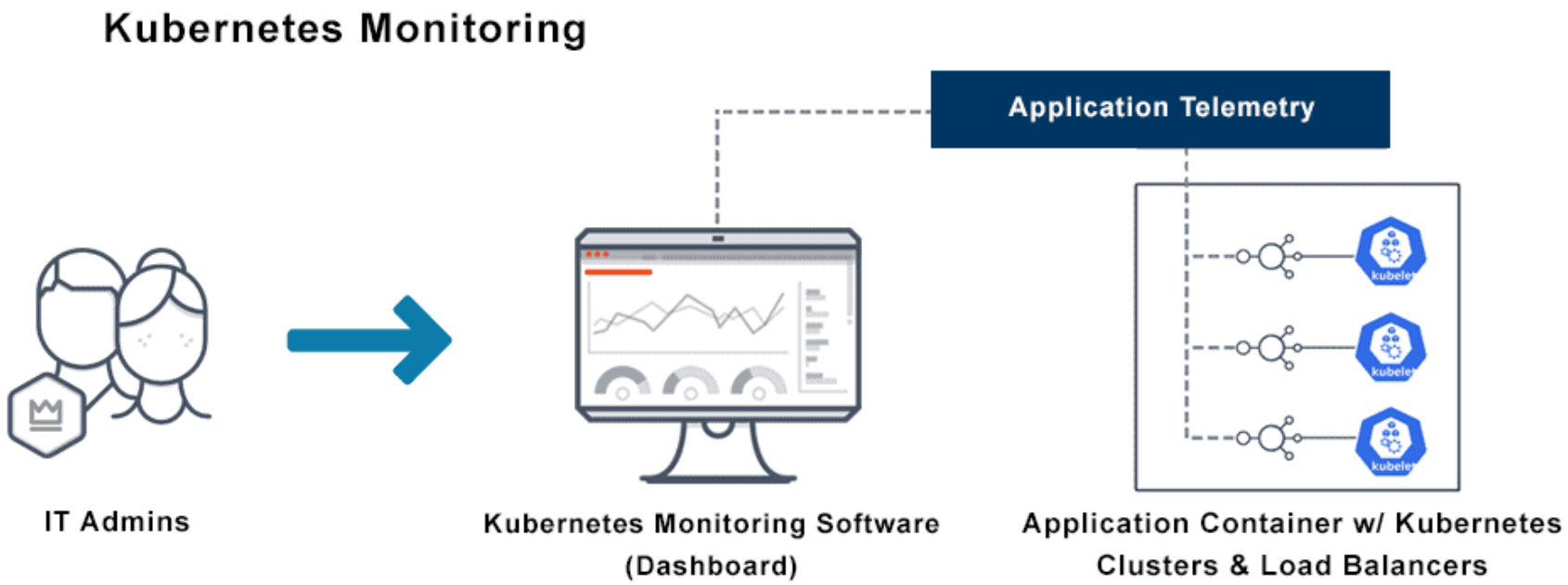


Three pillars of Observability

- Metrics
- Traces
- Logs



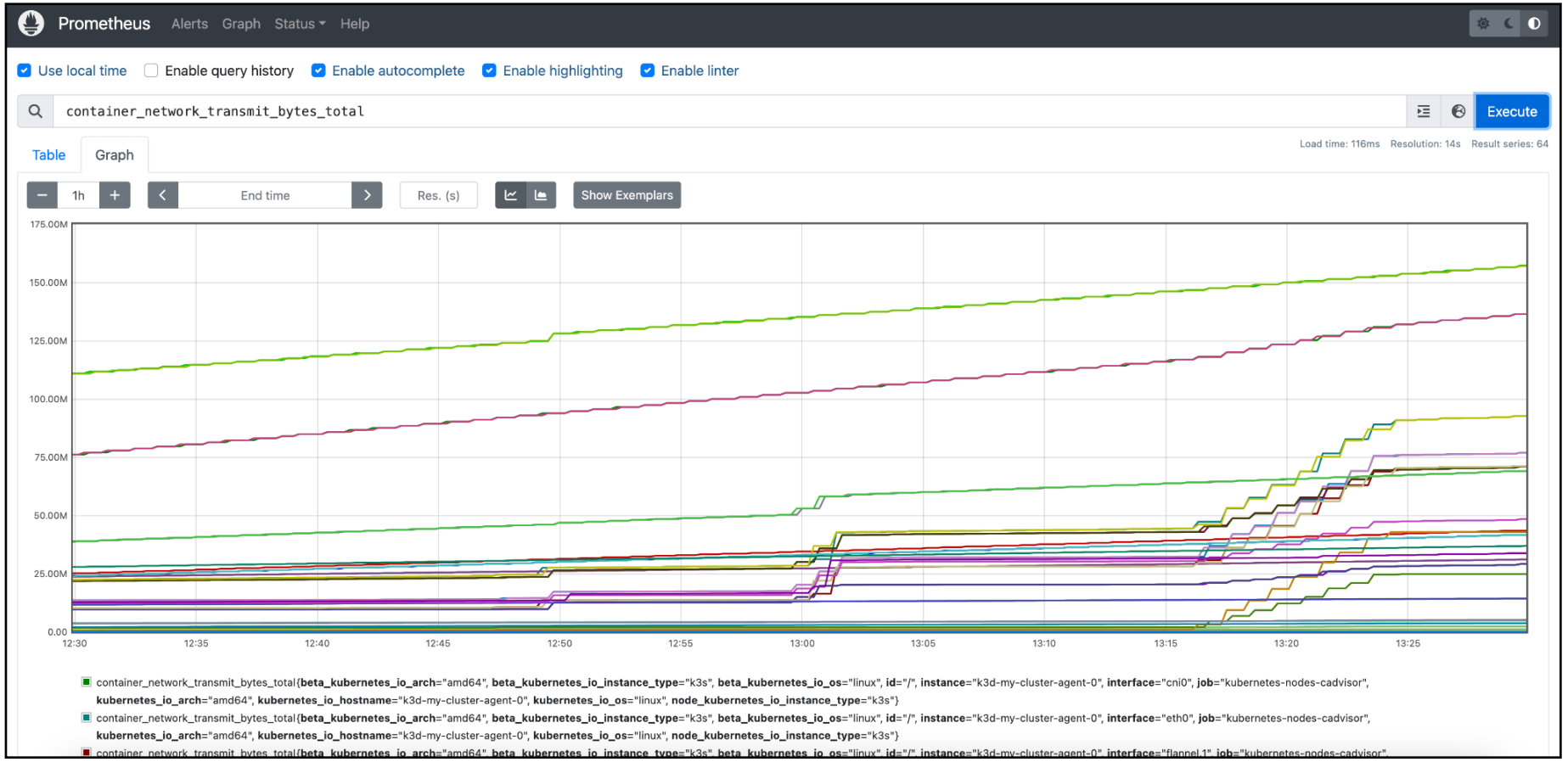
Metrics monitoring



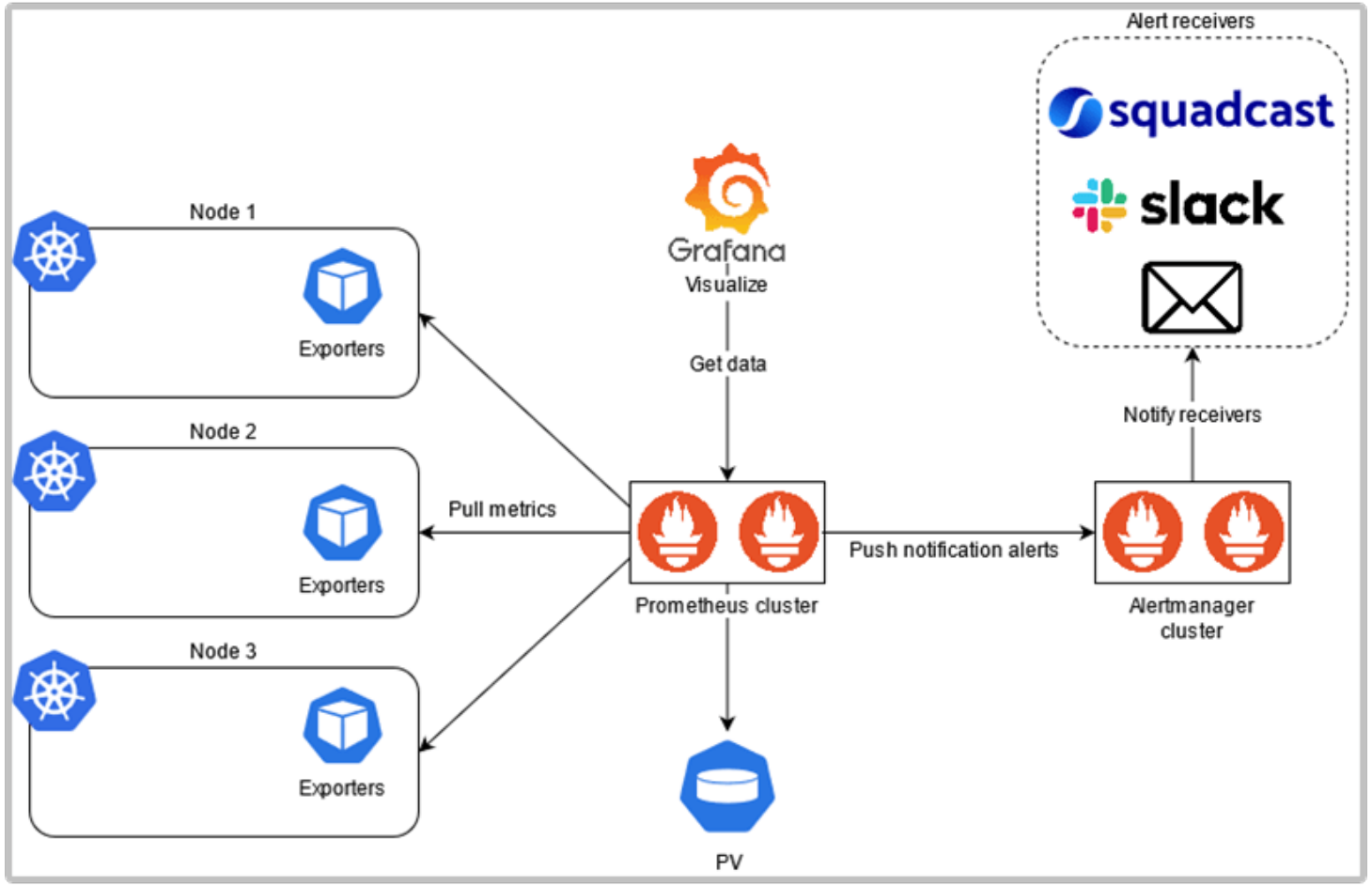
Metrics monitoring

- Cluster state metrics, including the availability and health of pods
- CPU utilization
- Disk utilization
- Node status, including disk or processor overload, memory, network availability, and readiness
- Pod availability (unavailable pods may indicate poorly designed readiness probes or configuration issues)

Prometheus



Prometheus alert



Prometheus alert example

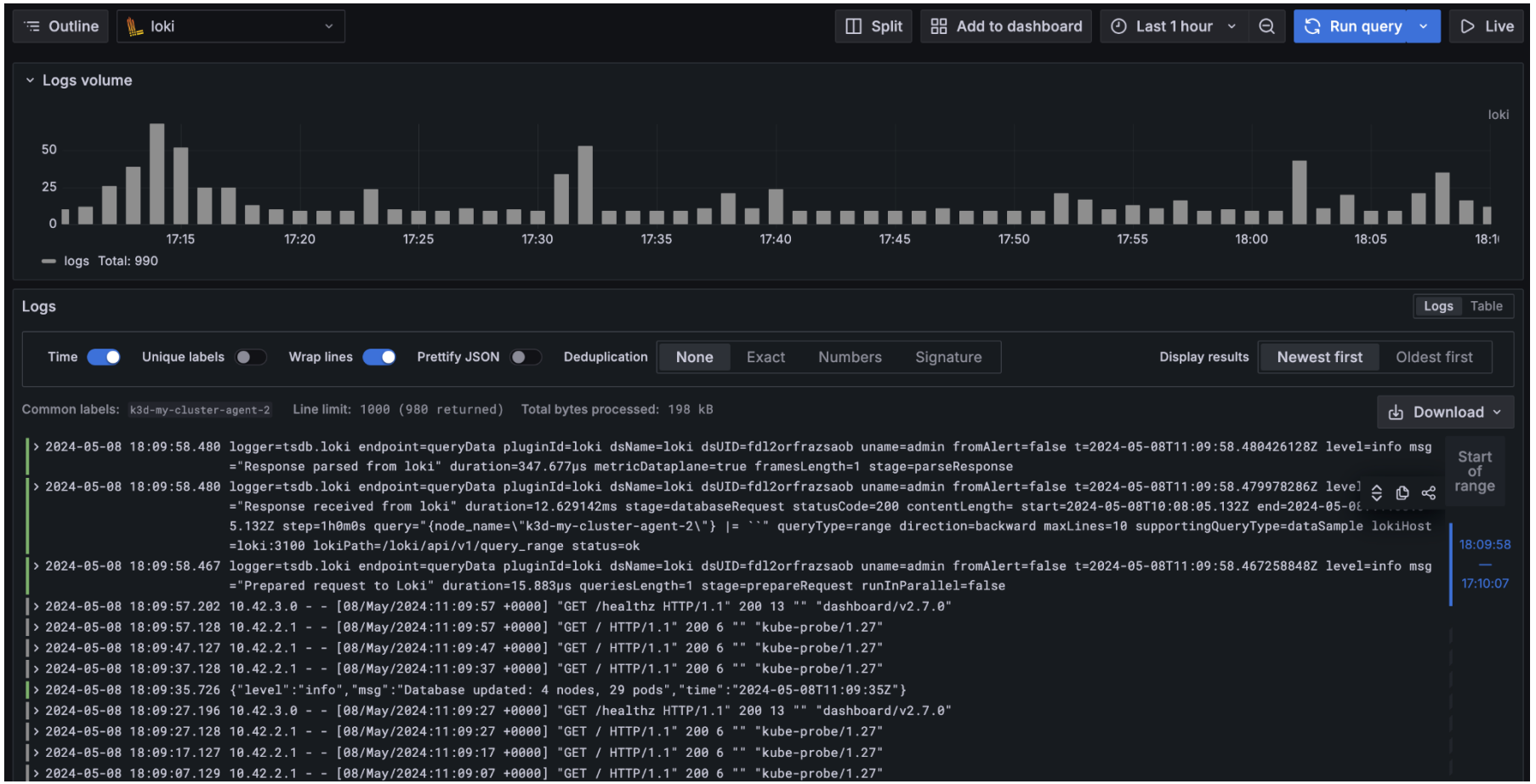
- https://samber.github.io/awesome-prometheus-alerts/rules

Grafana

Logs monitoring

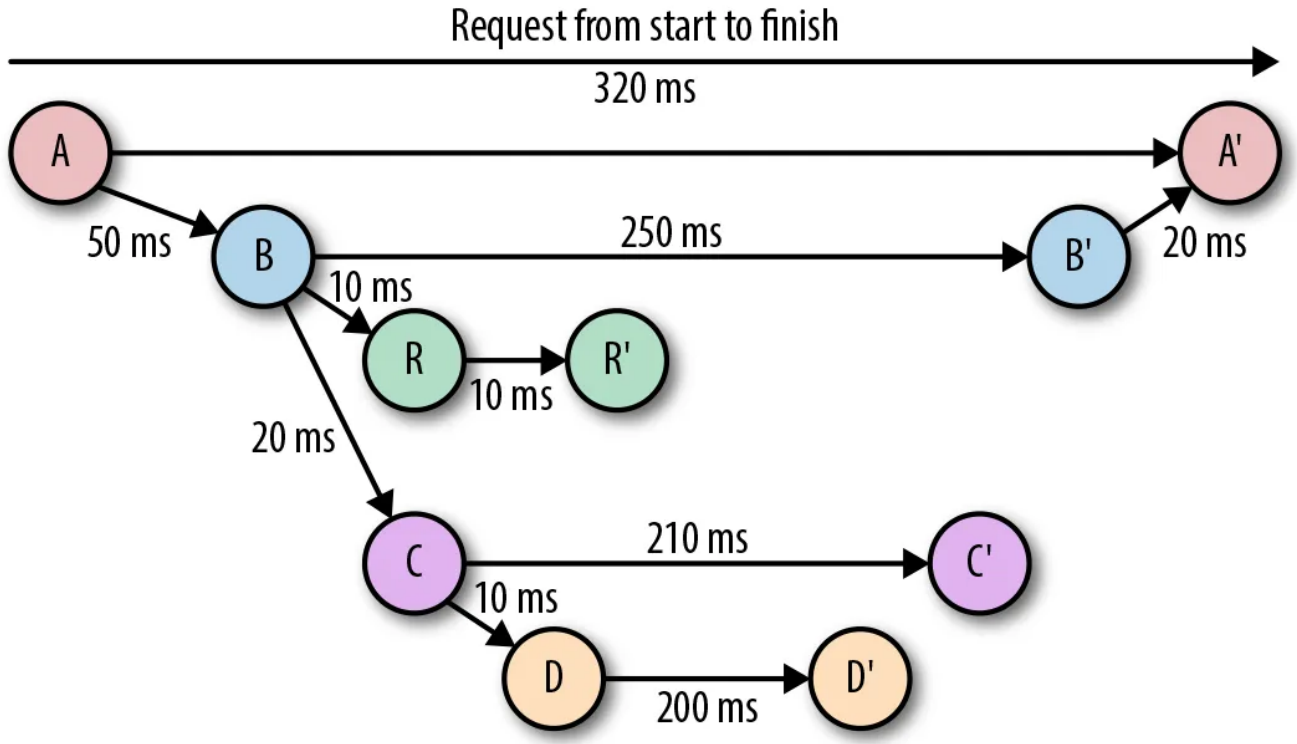
- Cluster logs
- Node logs
- Container logs
- Kubernetes Events

Loki



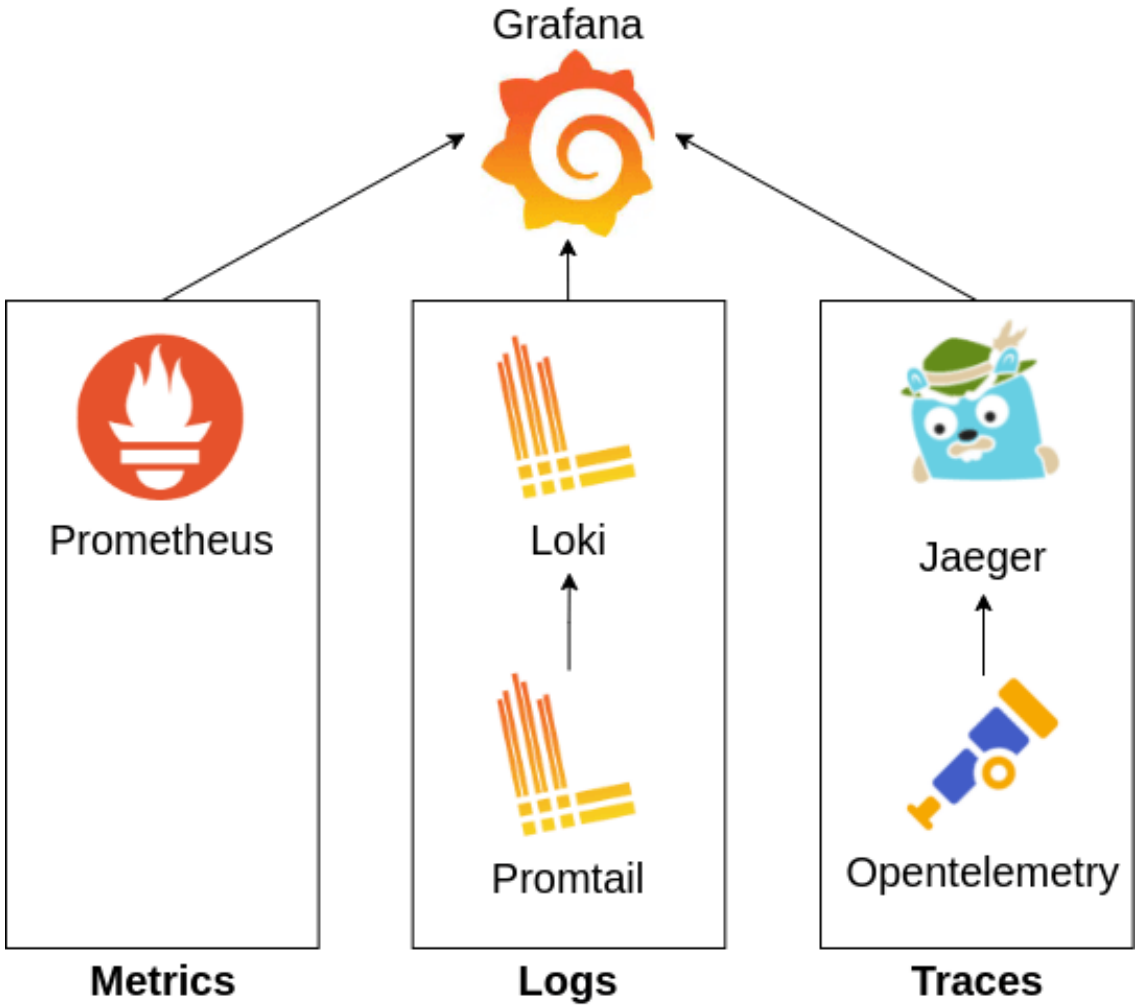
Traces monitoring

- A trace represents the entire journey of a request or action as it moves through all the nodes of a distributed system



Jaeger

Observability



Workshop Observability

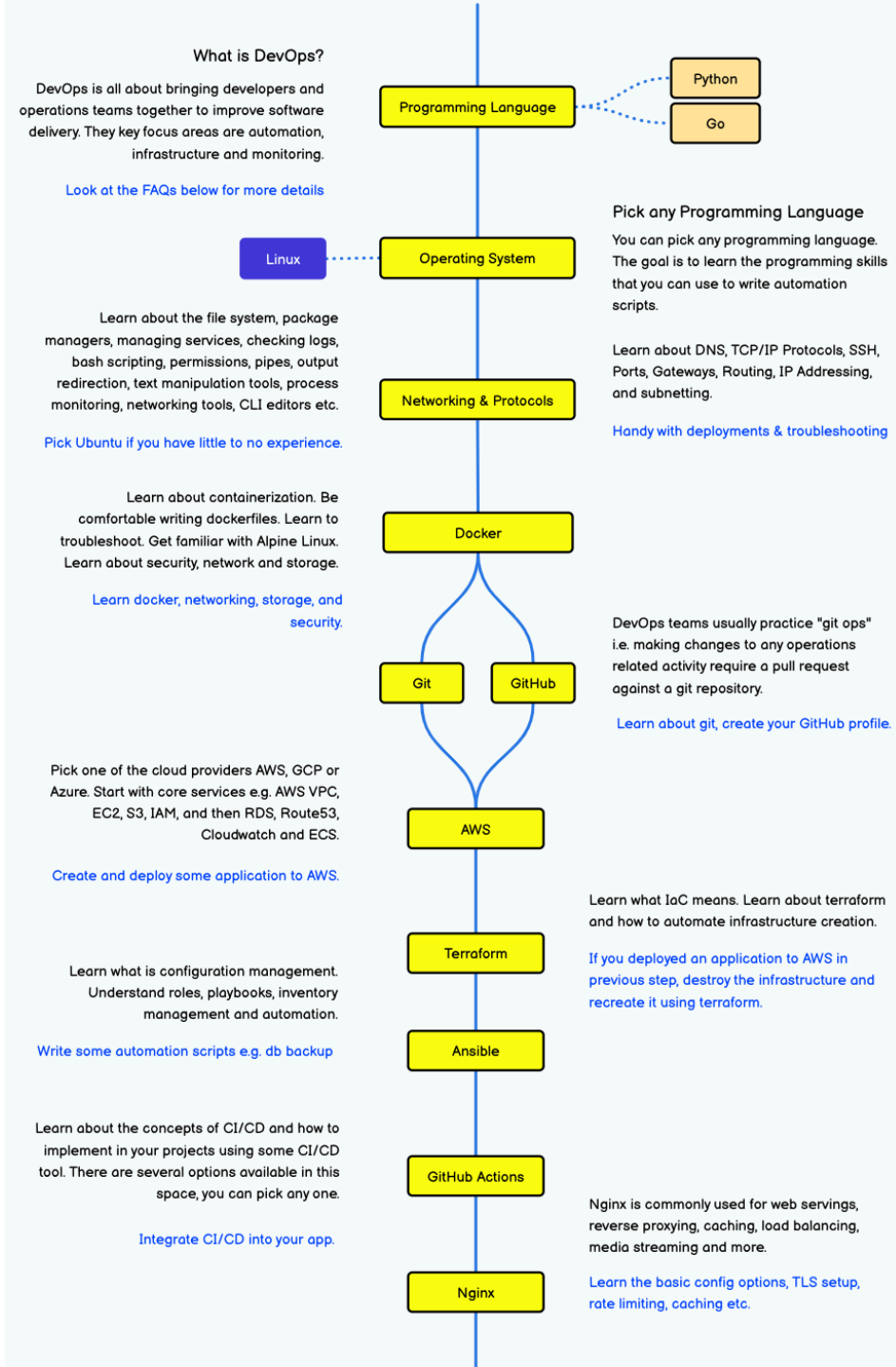
What next in DevOps?

- DevSecOps
- GitOps
- Artificial Intelligence and Machine Learning
- Serverless Computing
- Site Reliability Engineering

What is DevOps?

DevOps is all about bringing developers and operations teams together to improve software delivery. They key focus areas are automation, infrastructure and monitoring.

Look at the FAQs below for more details



Pick any Programming Language

You can pick any programming language. The goal is to learn the programming skills that you can use to write automation scripts.

Learn about the file system, package managers, managing services, checking logs, bash scripting, permissions, pipes, output redirection, text manipulation tools, process monitoring, networking tools, CLI editors etc.

Pick Ubuntu if you have little to no experience.

Learn about DNS, TCP/IP Protocols, SSH, Ports, Gateways, Routing, IP Addressing, and subnetting.

Handy with deployments & troubleshooting

Learn about containerization. Be comfortable writing dockerfiles. Learn to troubleshoot. Get familiar with Alpine Linux. Learn about security, network and storage.

Learn docker, networking, storage, and security.

DevOps teams usually practice "git ops" i.e. making changes to any operations related activity require a pull request against a git repository.

Learn about git, create your GitHub profile.

Pick one of the cloud providers AWS, GCP or Azure. Start with core services e.g. AWS VPC, EC2, S3, IAM, and then RDS, Route53, Cloudwatch and ECS.

Create and deploy some application to AWS.

Learn what IaC means. Learn about terraform and how to automate infrastructure creation.

If you deployed an application to AWS in previous step, destroy the infrastructure and recreate it using terraform.

Learn what is configuration management. Understand roles, playbooks, inventory management and automation.

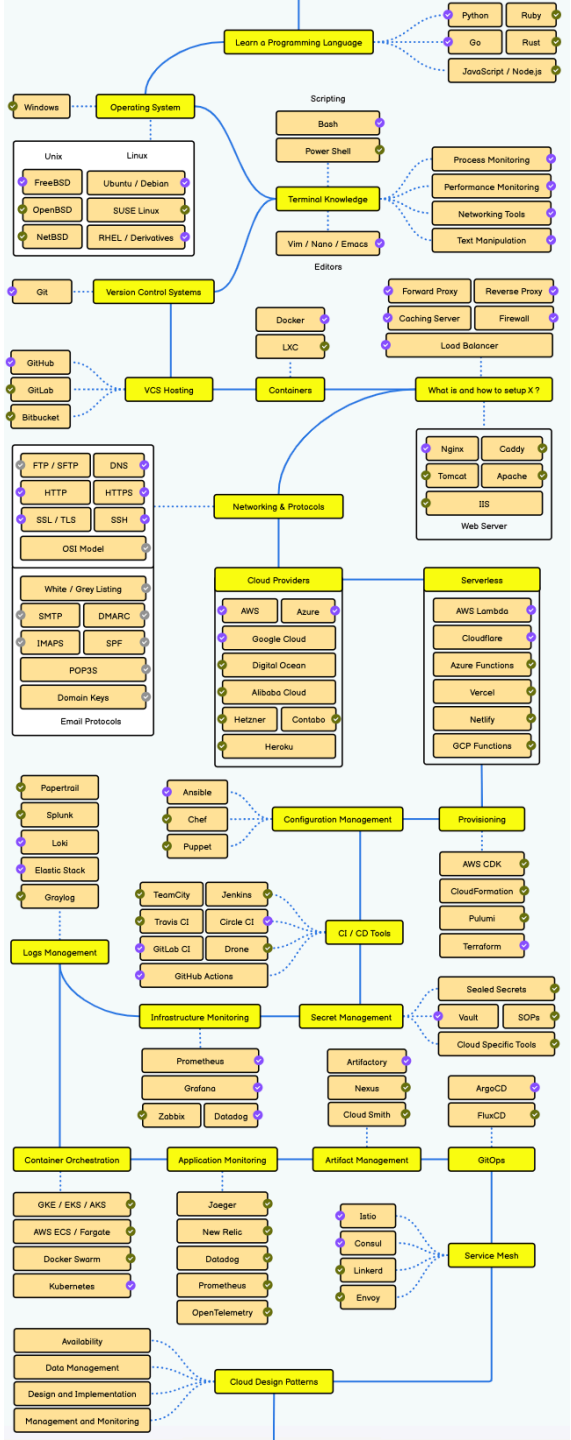
Write some automation scripts e.g. db backup

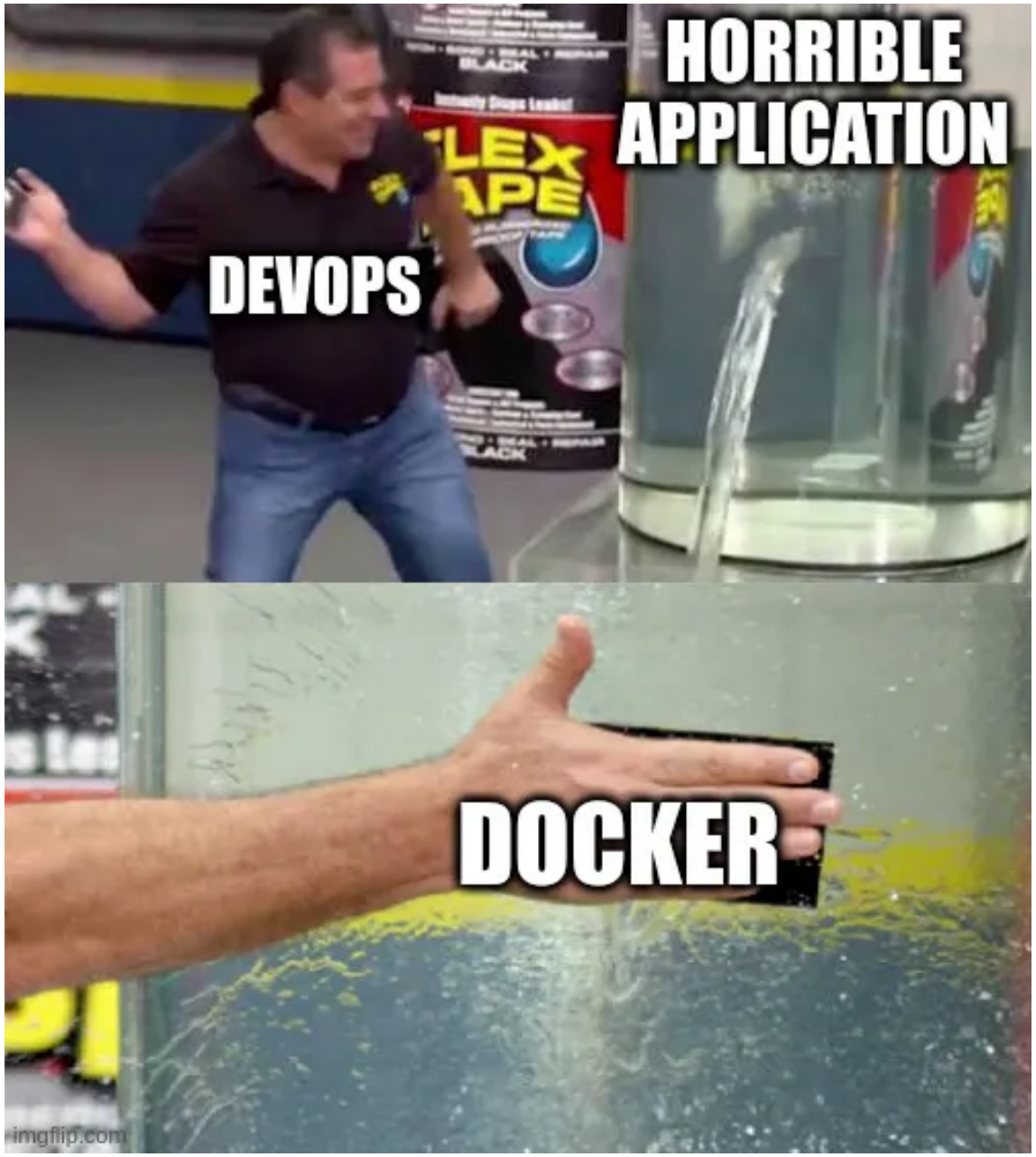
Learn about the concepts of CI/CD and how to implement in your projects using some CI/CD tool. There are several options available in this space, you can pick any one.

Integrate CI/CD into your app.

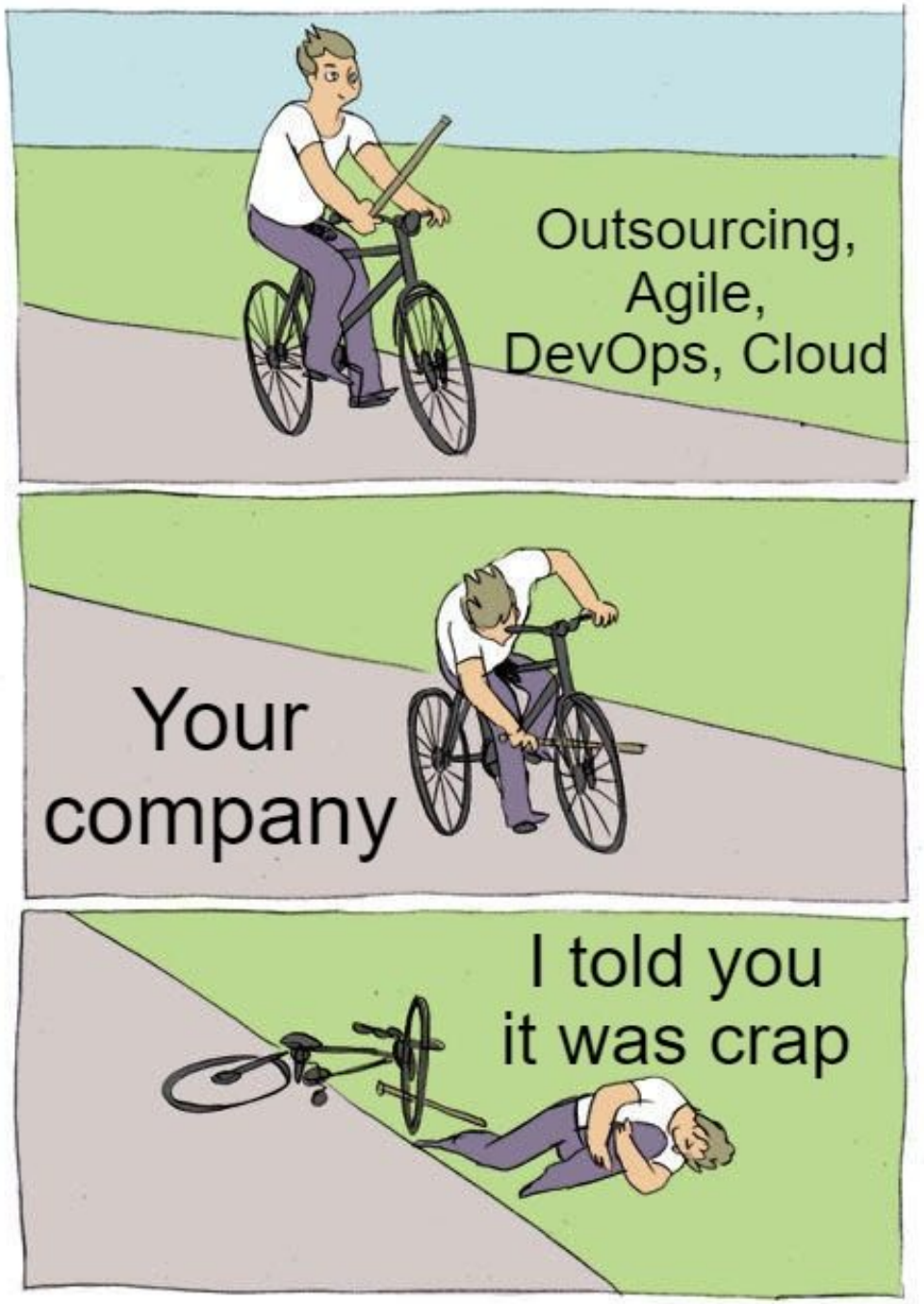
Nginx is commonly used for web servings, reverse proxying, caching, load balancing, media streaming and more.

Learn the basic config options, TLS setup, rate limiting, caching etc.





DevOps is not magic. It’s about **continuous daily improvement**. If it’s painful. You’re doing it wrong. Do it again, and **bring the pain forward**



3 C’s of Software Quality

- **Correctness**: perform its intended functions correctly and without errors or bugs
- **Completeness**: fulfills all its specified requirements. Ensures meets the needs of users
- **Consistency**: Ensures that the software is reliable and can be used in different situations.