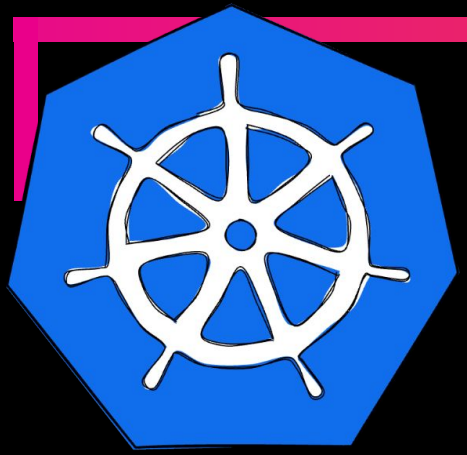
Welcome



introduction to

kubernetes

container orchestration system

DevelopIntelligence

A PLURALSIGHT COMPANY

Hello

About me...

- 15 years in industry
- 10 years in teaching
- Improv & Comedy expert
- Currently learning card magic
- Dvorak keyboard user

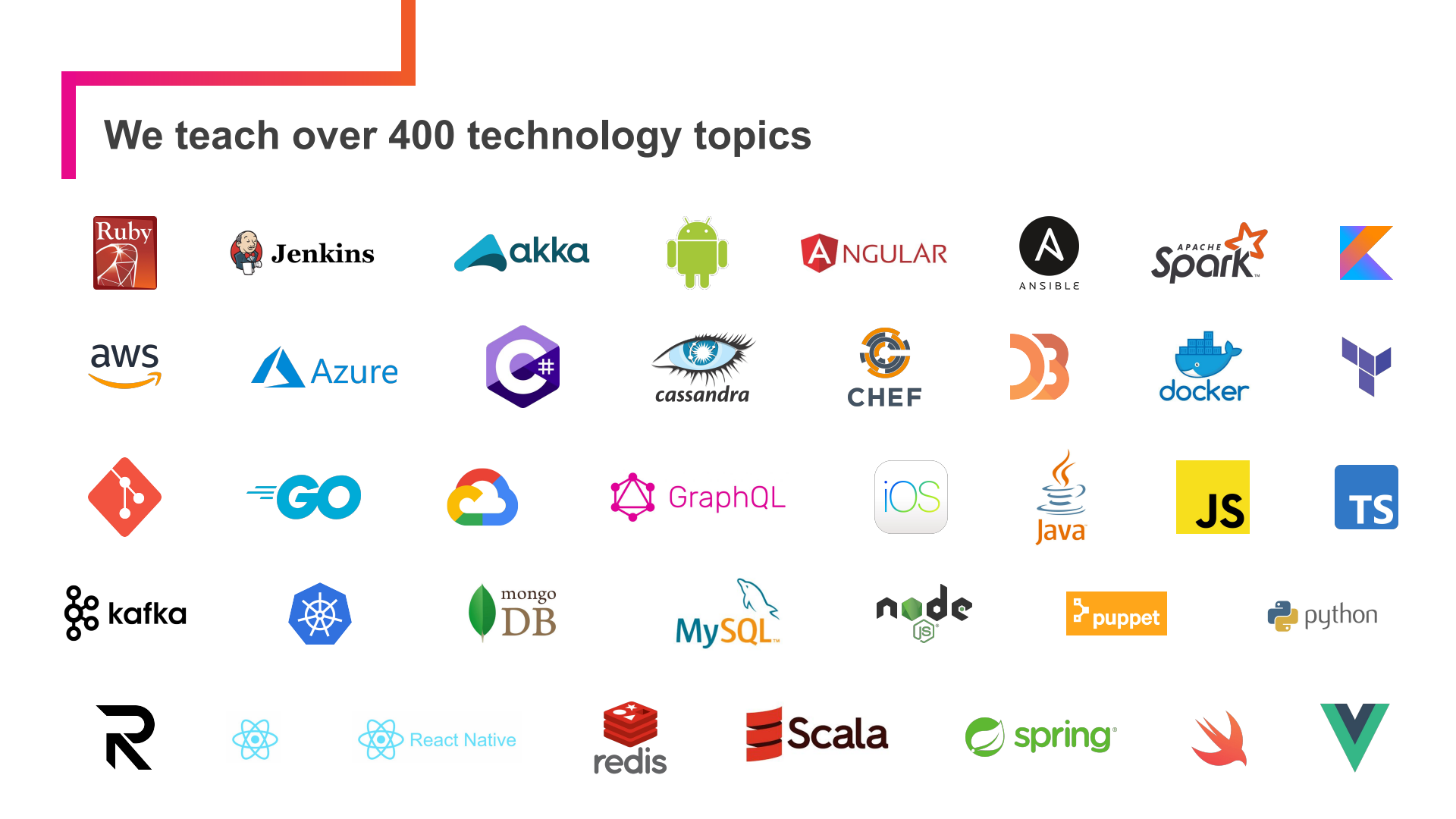
Prerequisites

This course assumes:

- You have a working knowledge of 1 or more programming languages

- This course is designed for existing software developers seeking to learn more about Kubernetes and how to use it

We teach over 400 technology topics

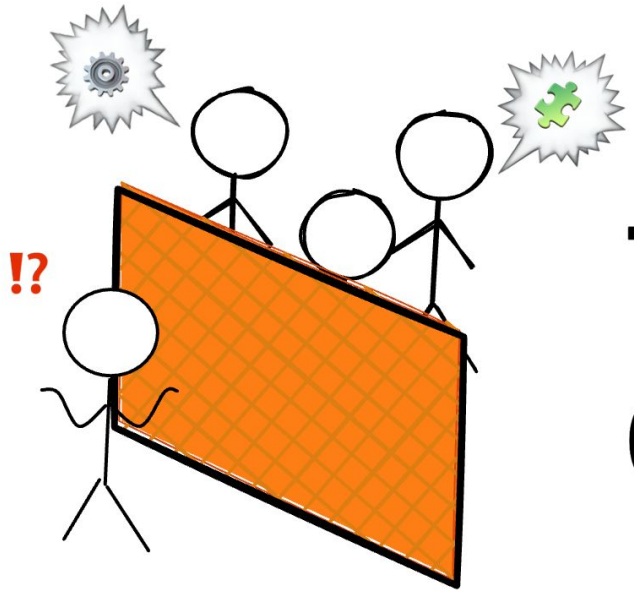


You experience our impact on a daily basis!

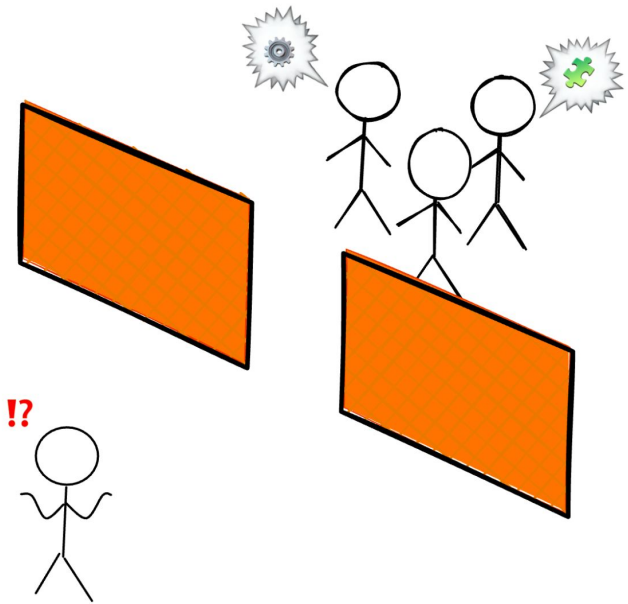
My pledge to you

I will...

- Make this interactive
- Ask you questions
- Ensure everyone can speak
- Use an on-screen timer
- ***Model good documentation-reading***

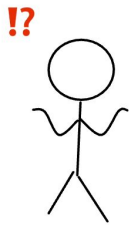


Threshold Concept

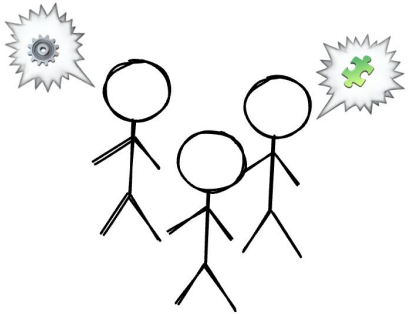


Let's start by prying that wall open a bit.

Creating a portal to the other side.

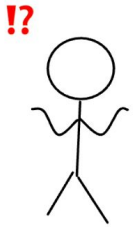


From the top-down

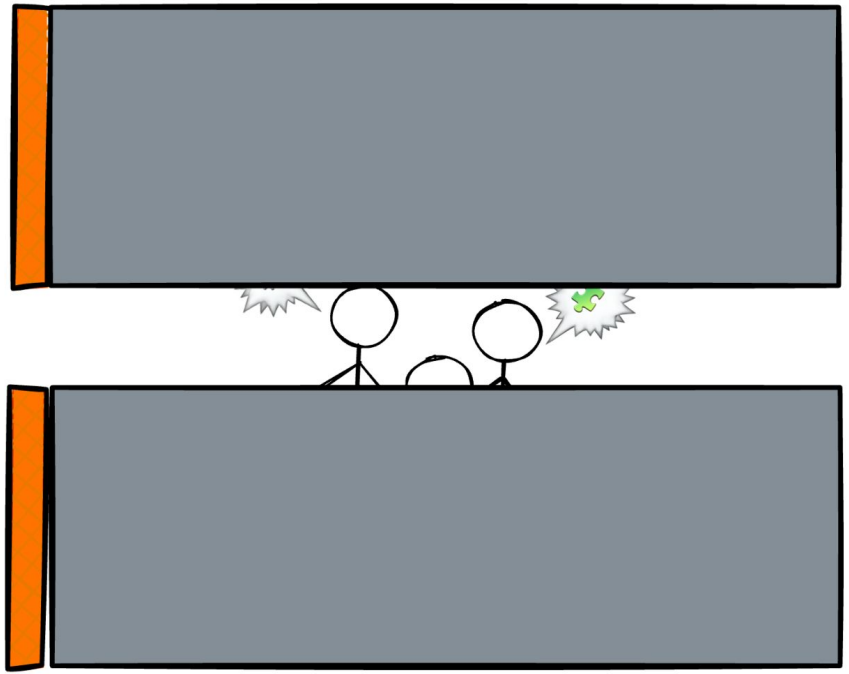


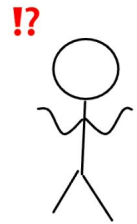
it looks a little bit like this.

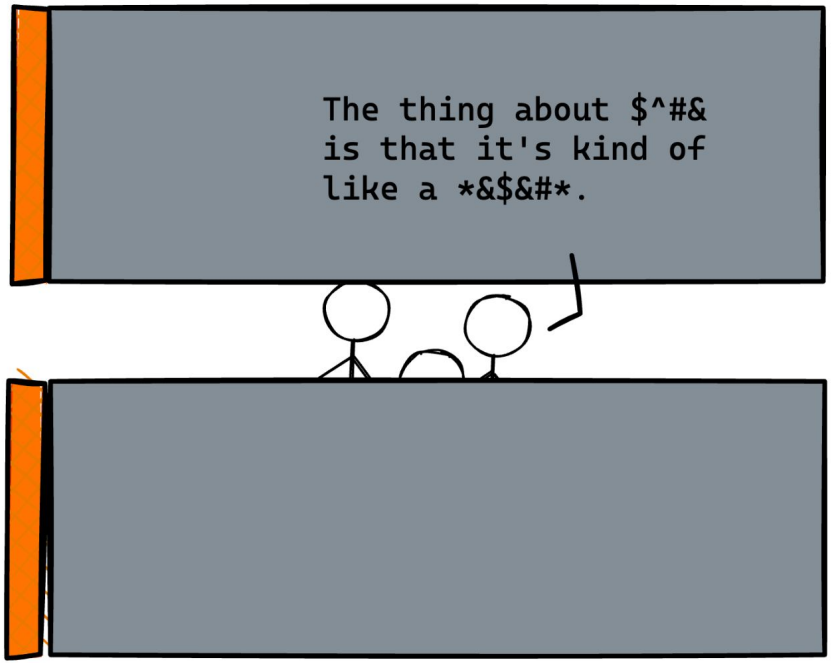
But really, it's actually like this.



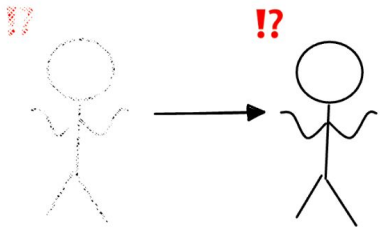
From so far away, your view across the threshold is constrained.

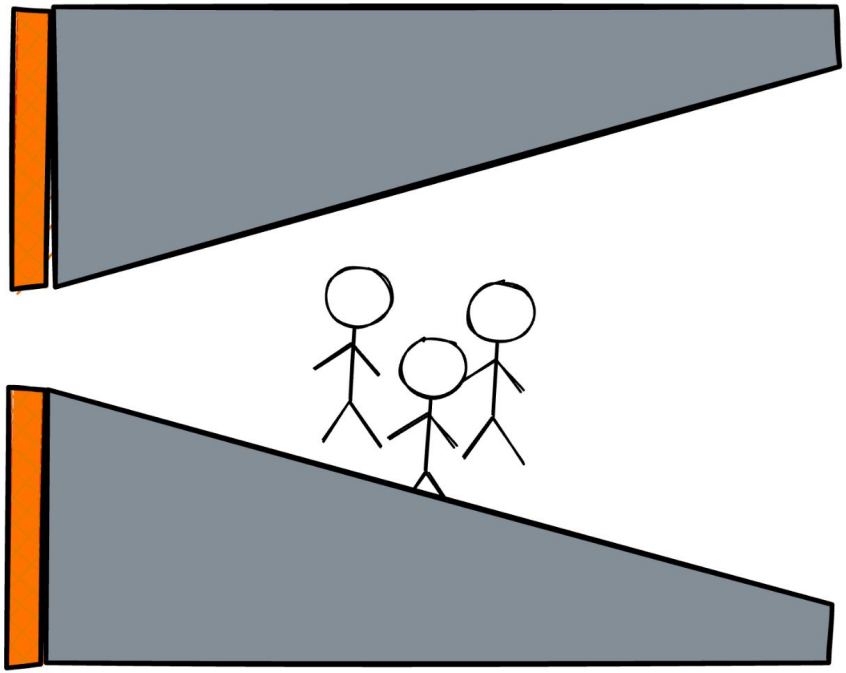



!?

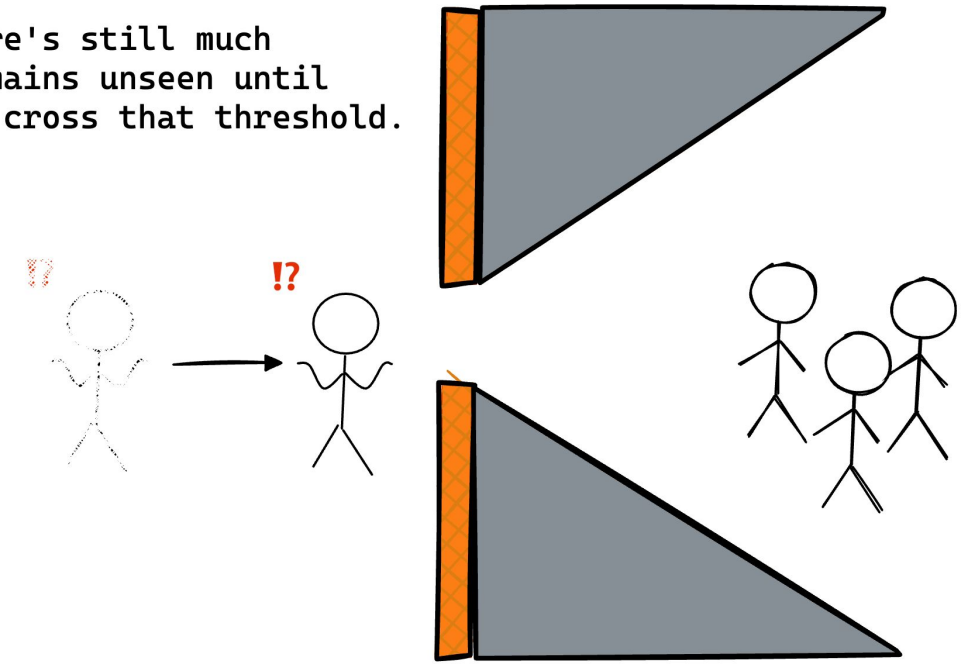

The thing about $^#&
is that it's kind of
like a *&$&#*.

If you can get closer, you can see a little more.





But there's still much
that remains unseen until
you can cross that threshold.



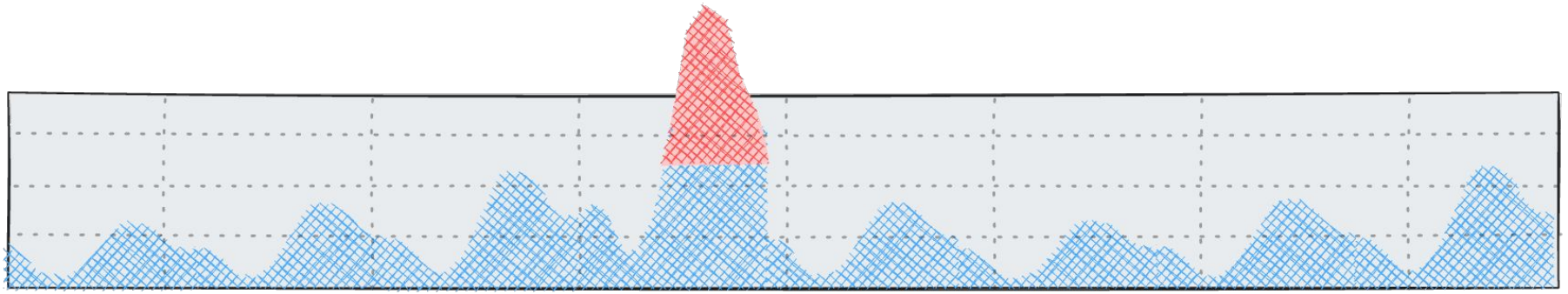
Agenda

- Kubernetes and container orchestration
- How do Docker and K8s fit together?
- Diving deeper into several Kubernetes primitives including Pods, Deployments, Services, ConfigMaps, Secrets, etc.
- Discussion of key areas of focus like configuration, observability, and persistence
- The architecture of a k8s cluster

How we're going to work together

- Slides / lecture
- Demos
- Discussions
- Labs

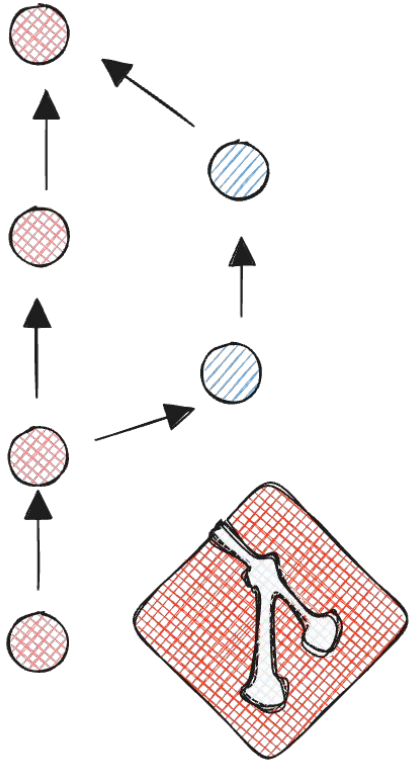
Software Deployment Challenges

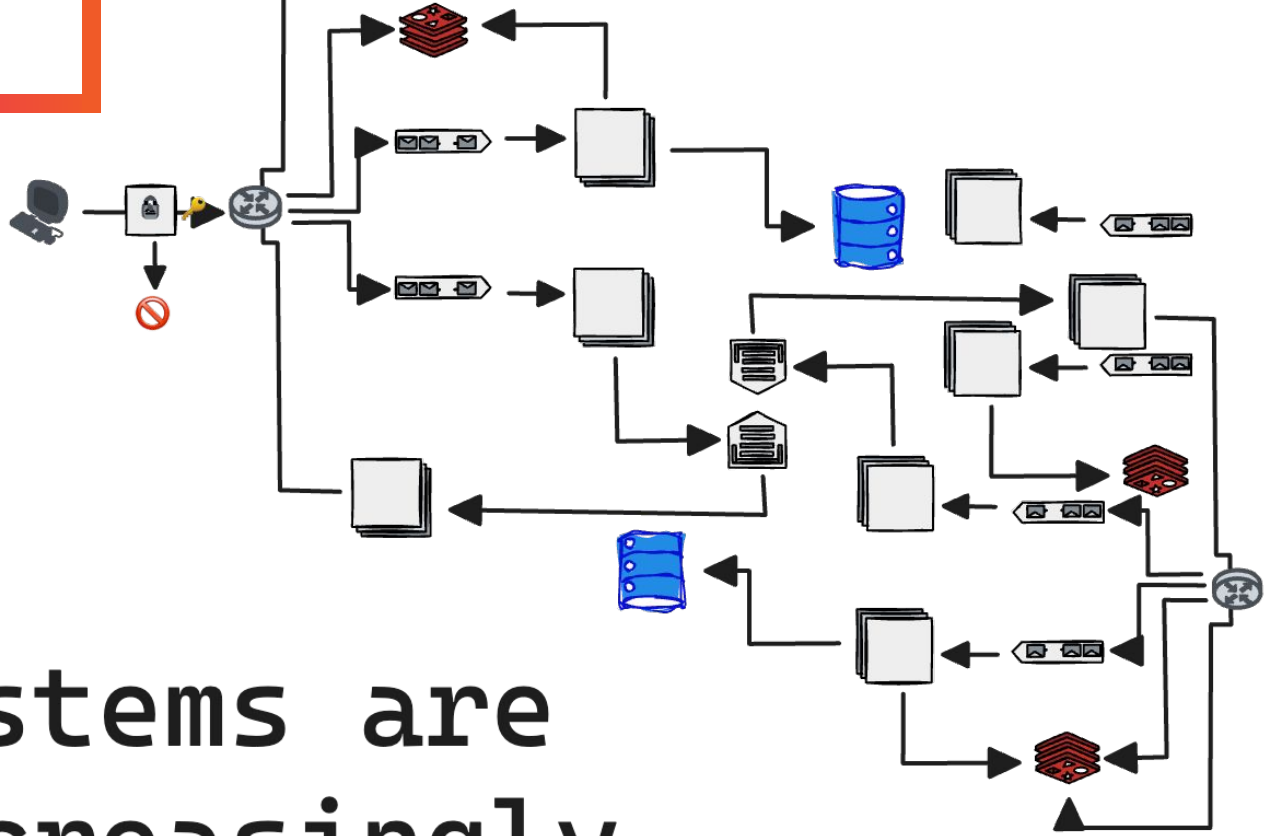


Demand is elastic

Software Deployment Challenges

Changes are frequent

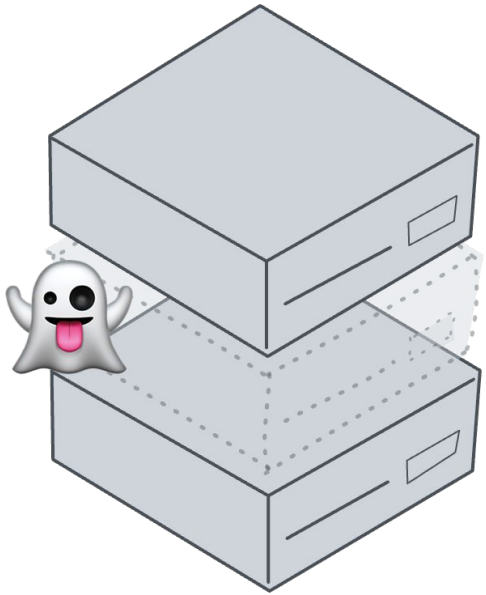




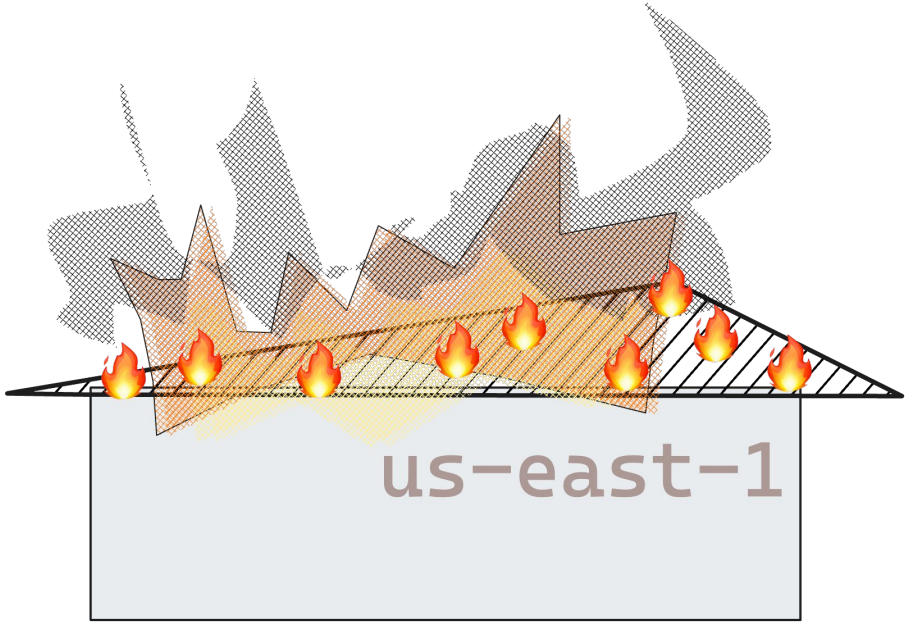
Systems are increasingly complex

Software Deployment Challenges

The parts are unreliable

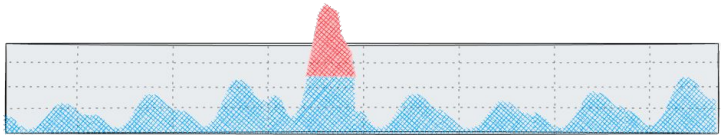


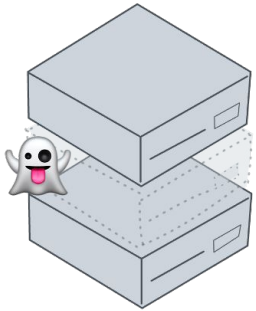
Software Deployment Challenges

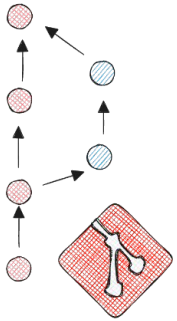


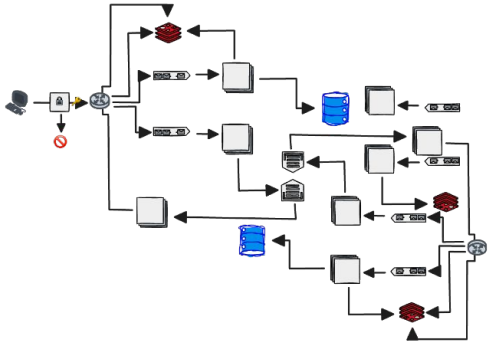
The data centers catch on fire.

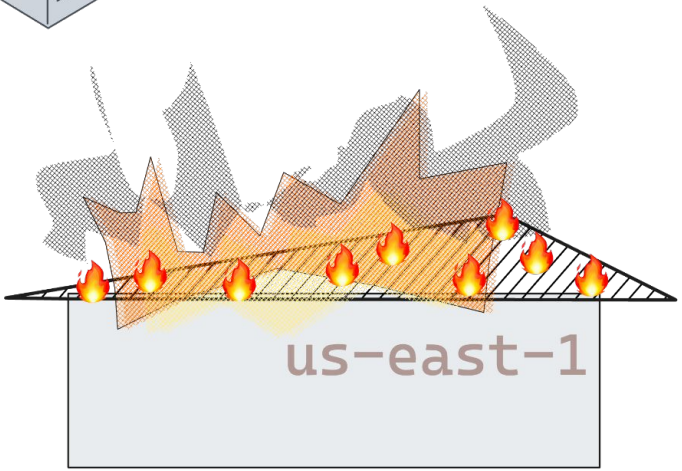
Software Deployment Challenges







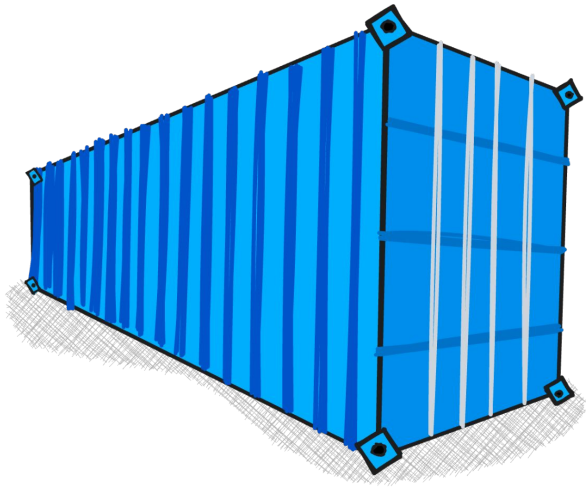




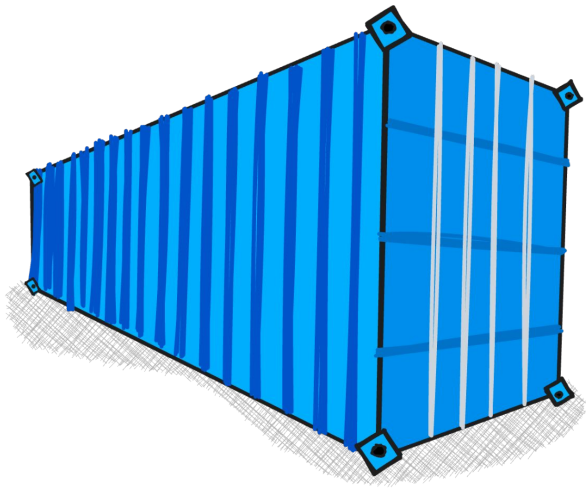
Containerization as a Deployment Strategy

Application Hosting

By Application Hosting, we mean the target infrastructure and runtime platform used for deployment and execution of an application or system; can include compute (CPU and server resources), storage, network, data and operating system



shipping container analogy



Isolated
Consistent
Repeatable
Portable
Scalable

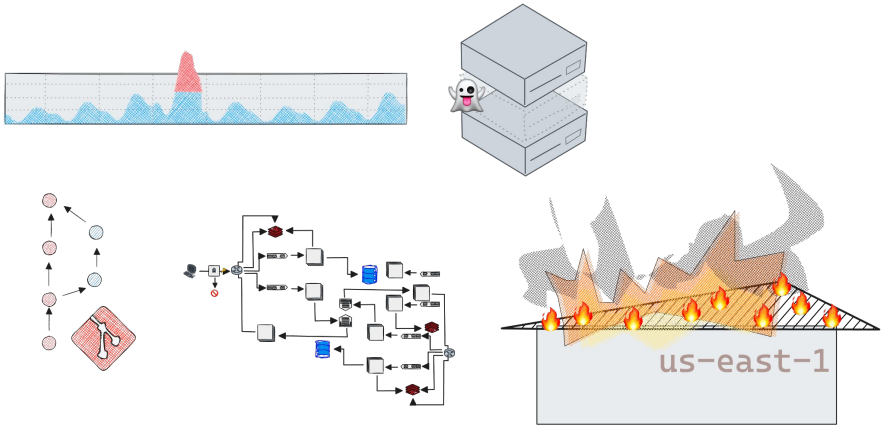
Isolated
Consistent
Repeatable
Portable
Scalable

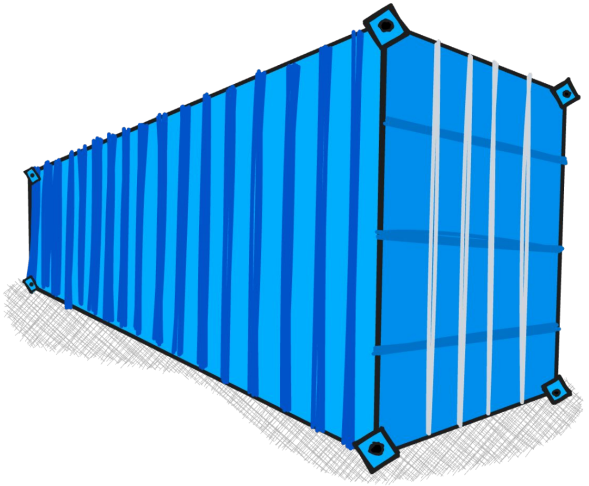
Isolated
Consistent
Repeatable
Portable
Scalable

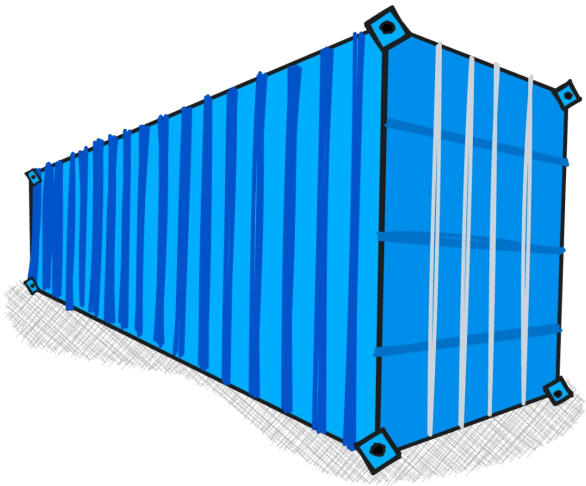
Isolated
Consistent
Repeatable
Portable
Scalable

Software Deployment Challenges

Problem ⟶ Solution



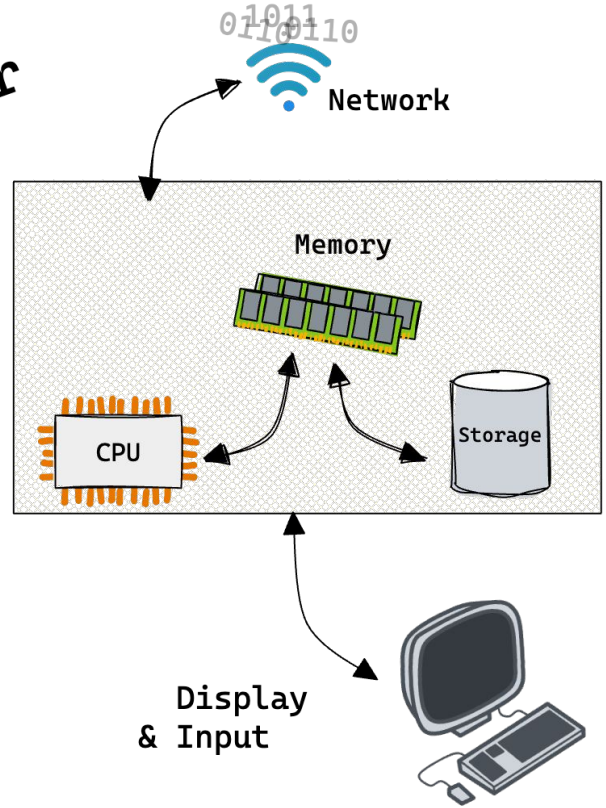




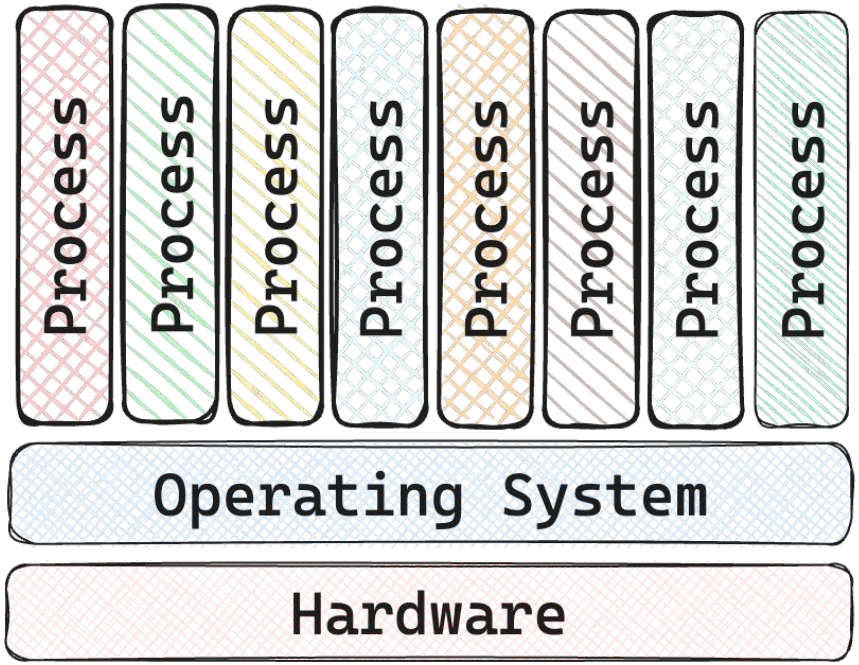
New questions

- How do we make a container?
- What goes on inside?
- What does the "port" do with them?

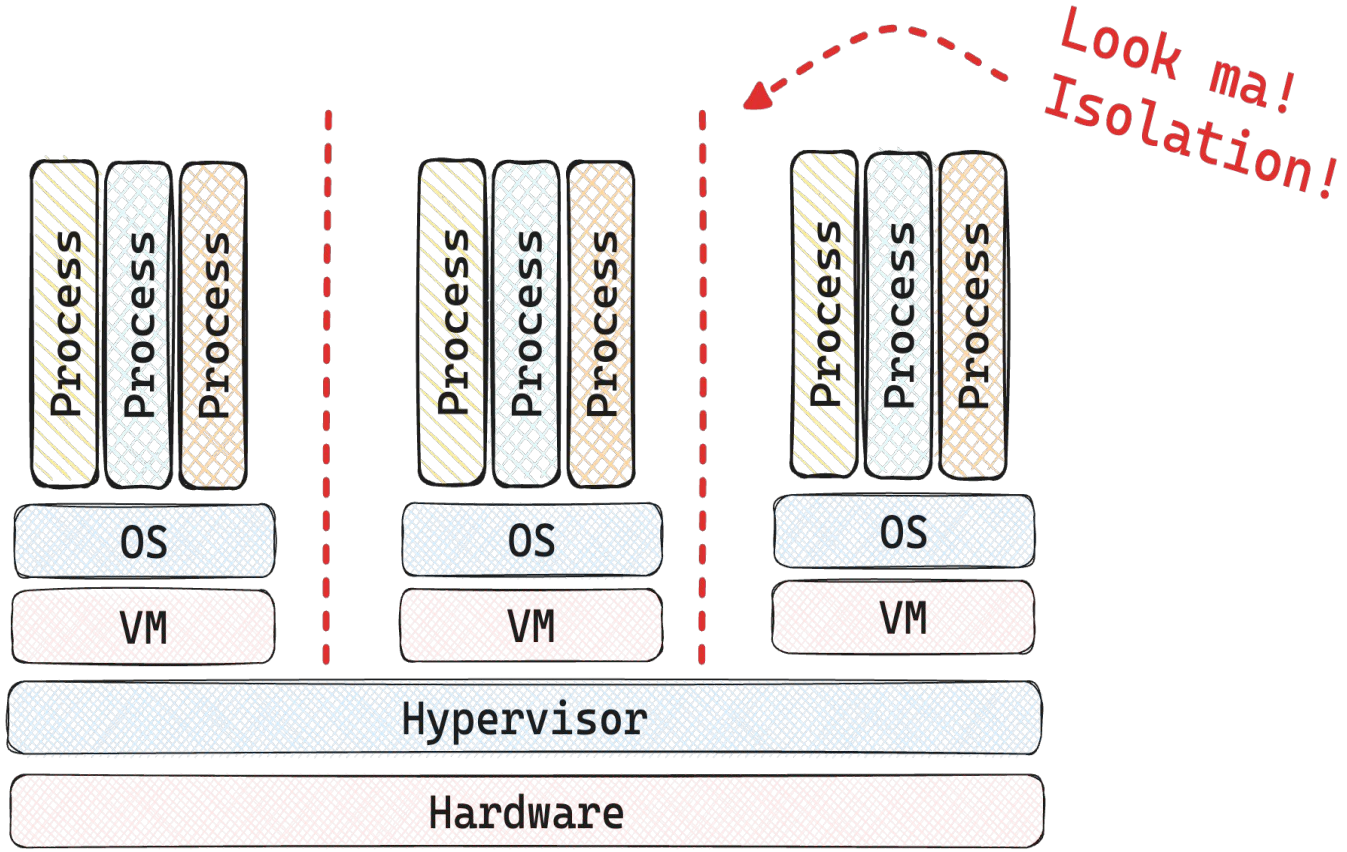
Rough Model of a Computer

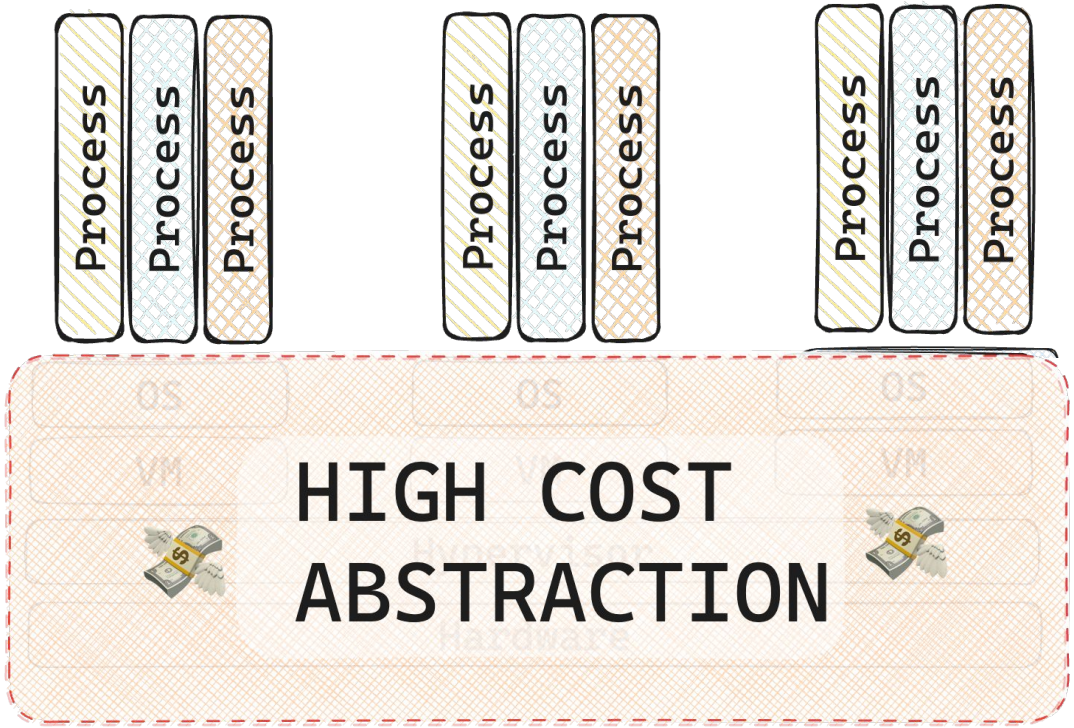


Rough Model of System Architecture

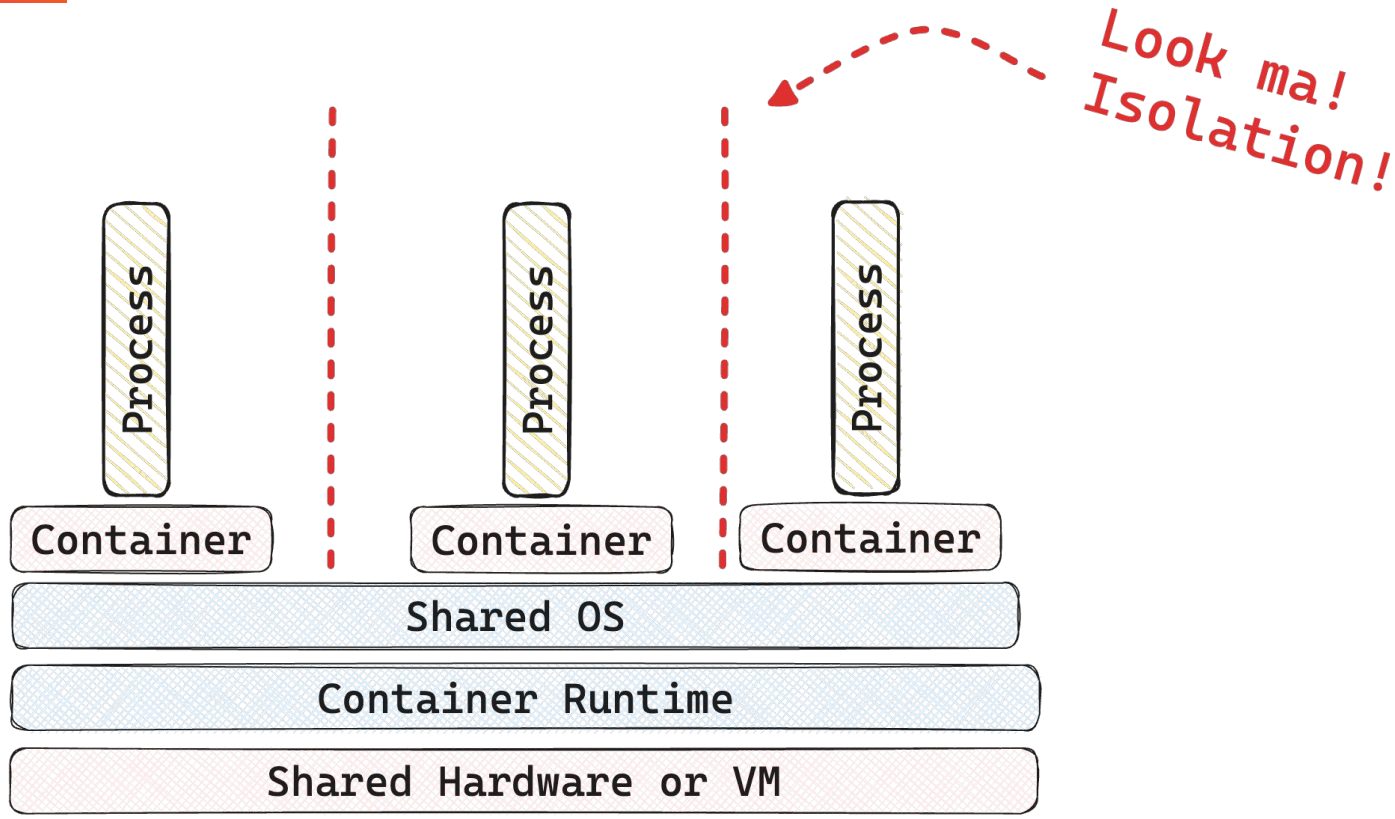


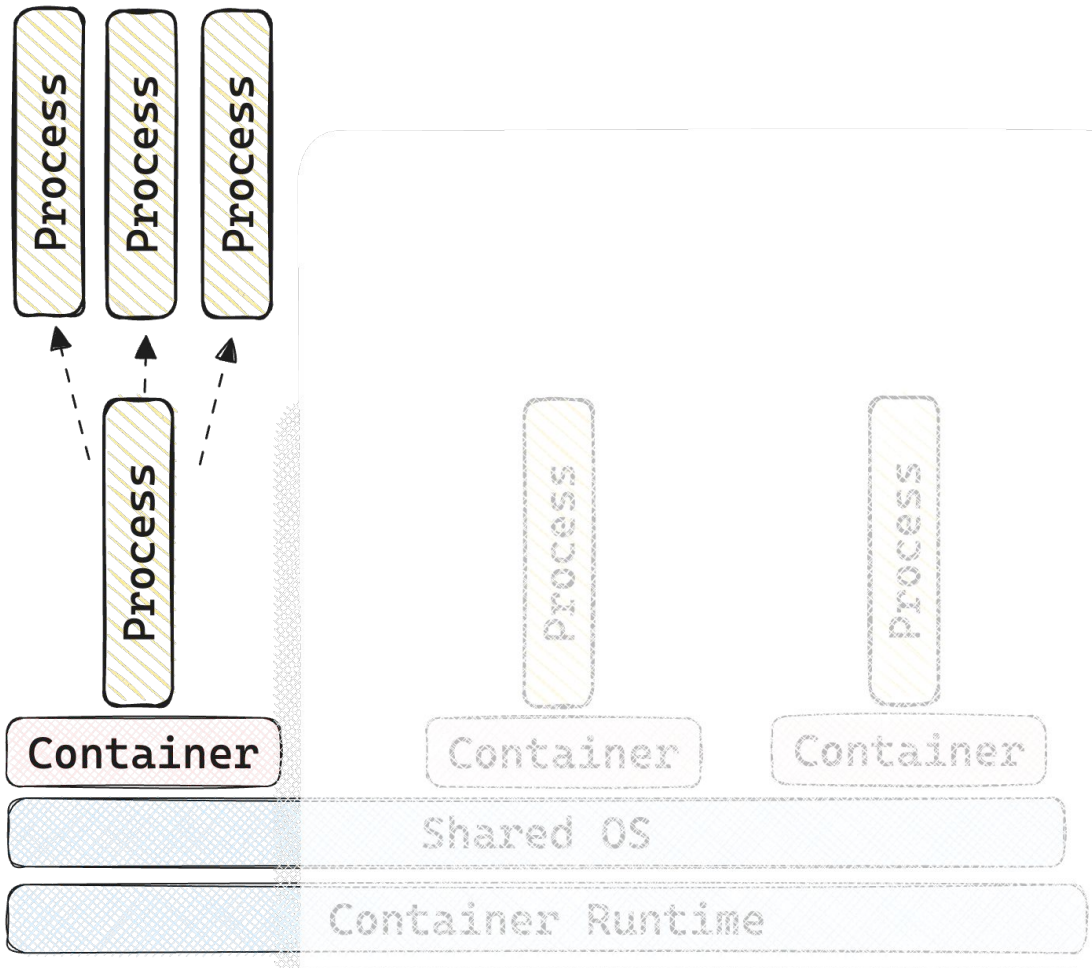
Rough Model of Virtual Machines



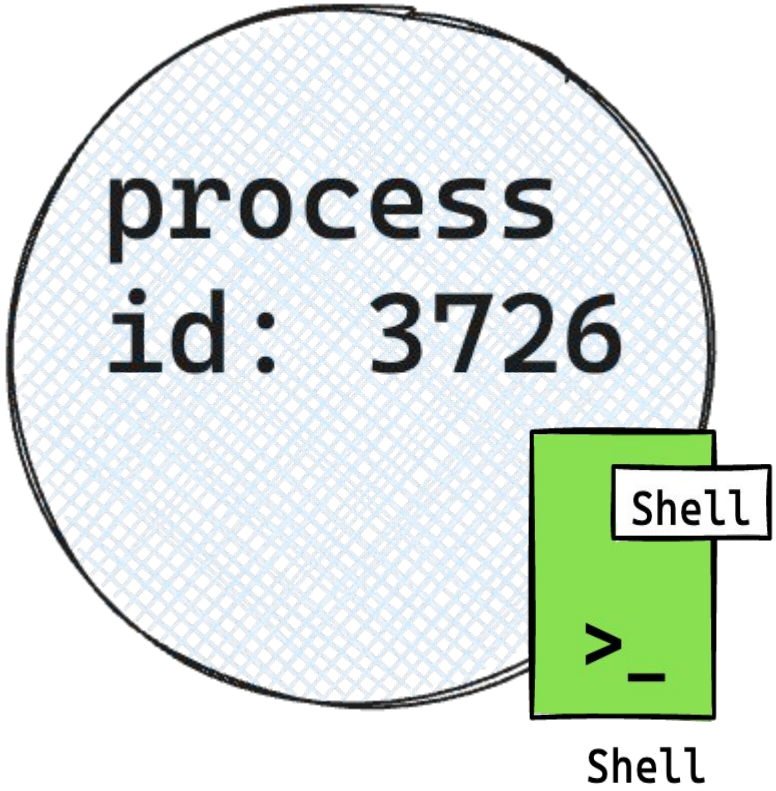

Process
Process
Process
Process
Process
Process
Process
Process
Process
OS
OS
OS
VM
VM
Hardware
HIGH COST
ABSTRACTION

Rough Model of Container Runtime

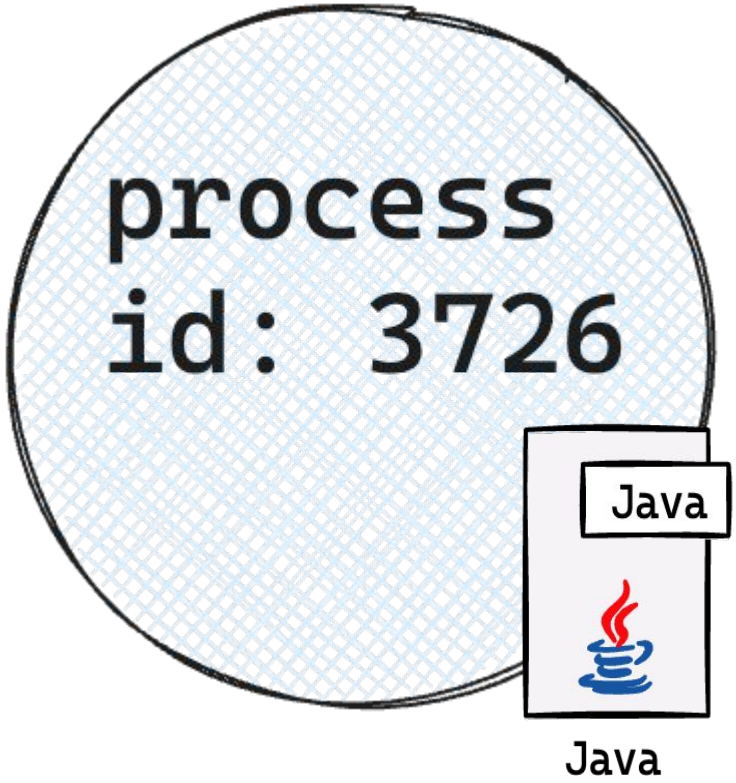



Process
Process
Process
Process
Container
Process
Container
Process
Container
Shared OS
Container Runtime

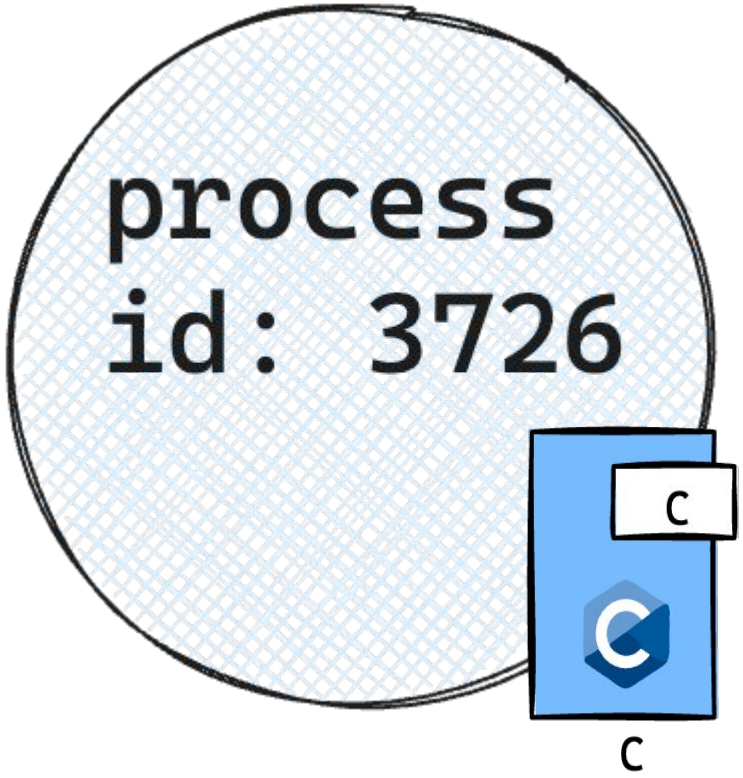
Rough Model of a Process



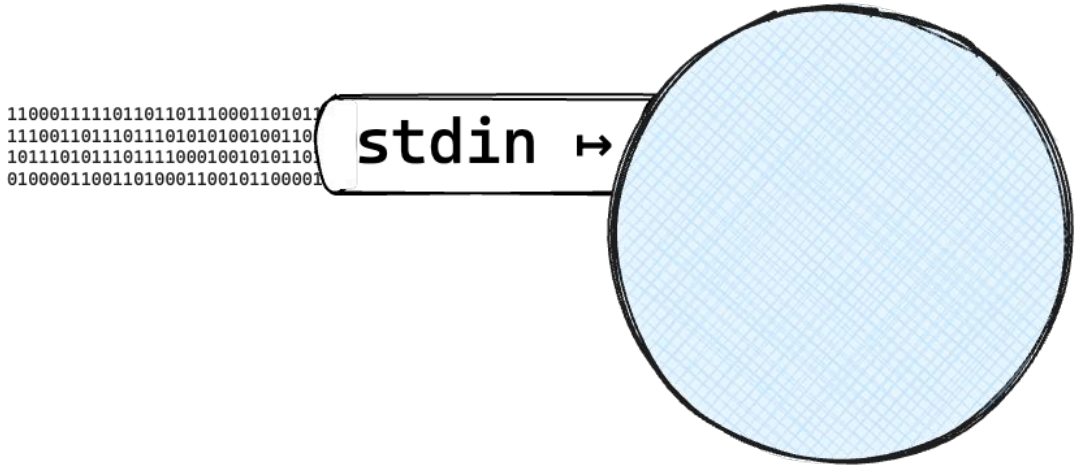
Rough Model of a Process



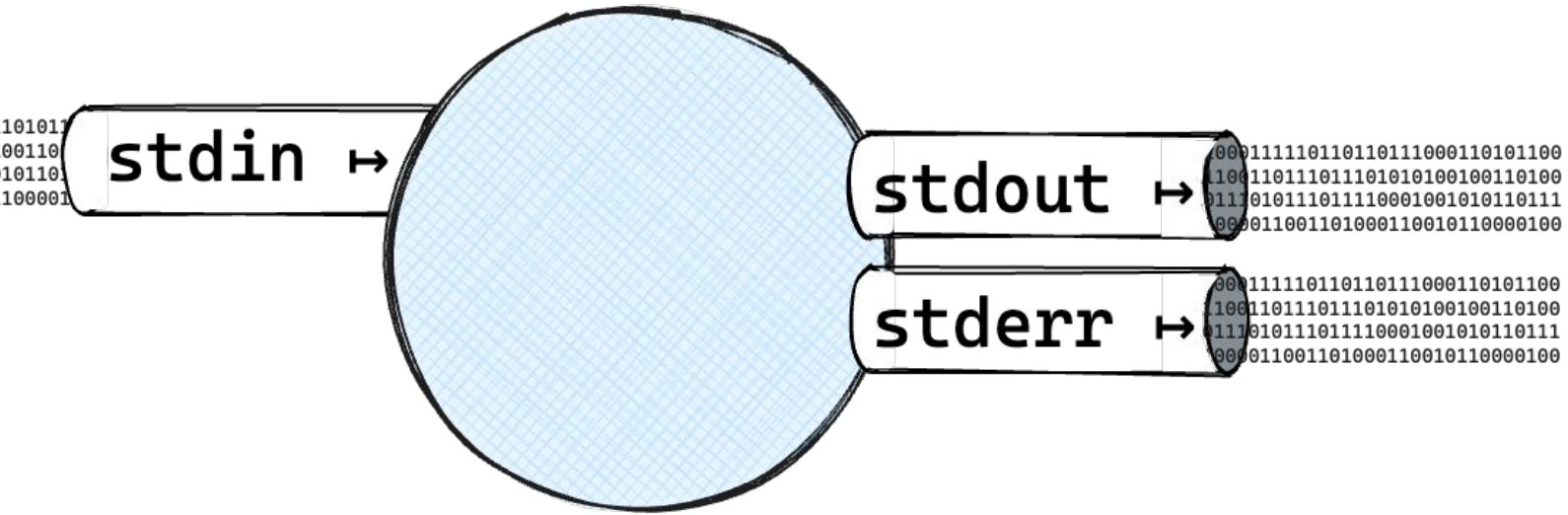
Rough Model of a Process



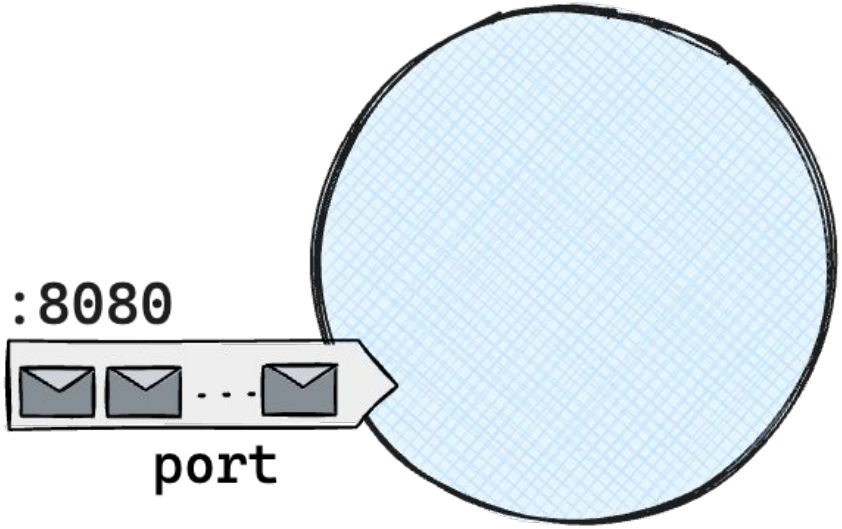
Rough Model of a Process



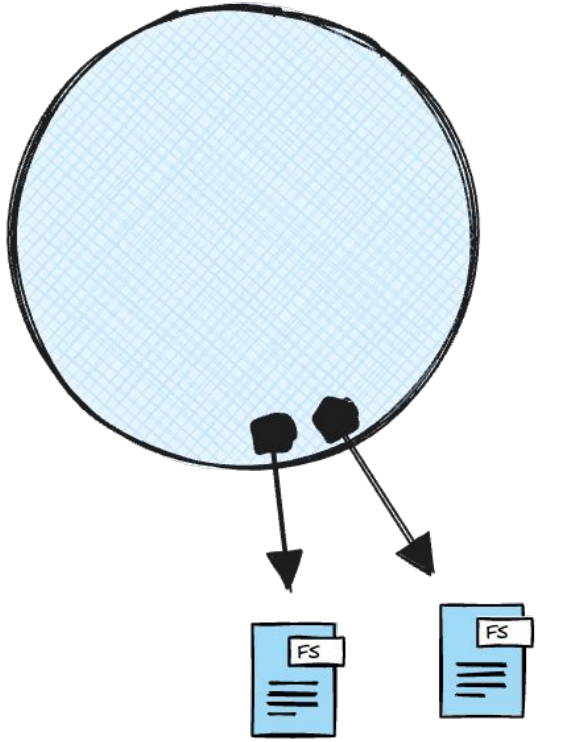
Rough Model of a Process



Rough Model of a Process

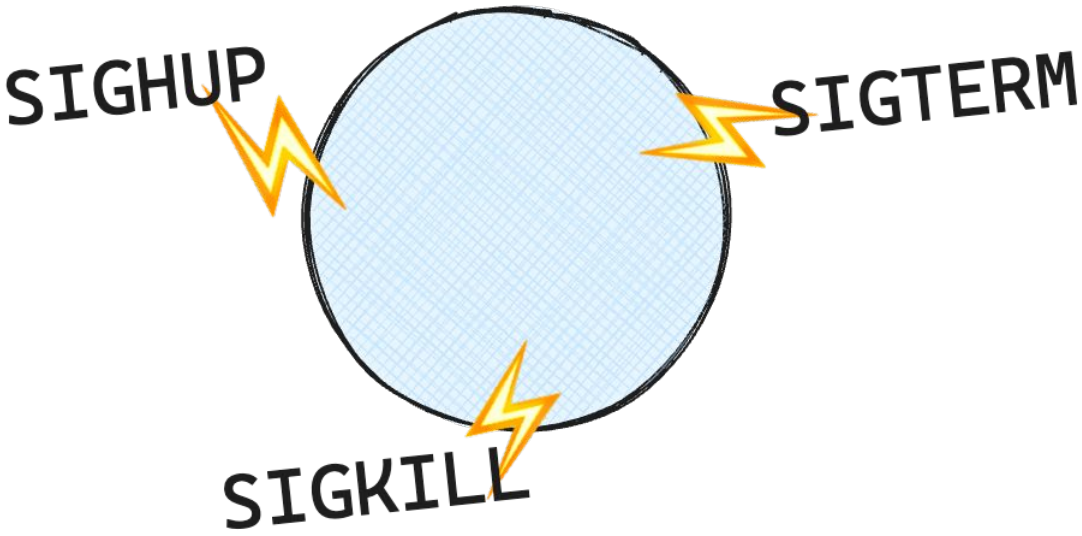


Rough Model of a Process

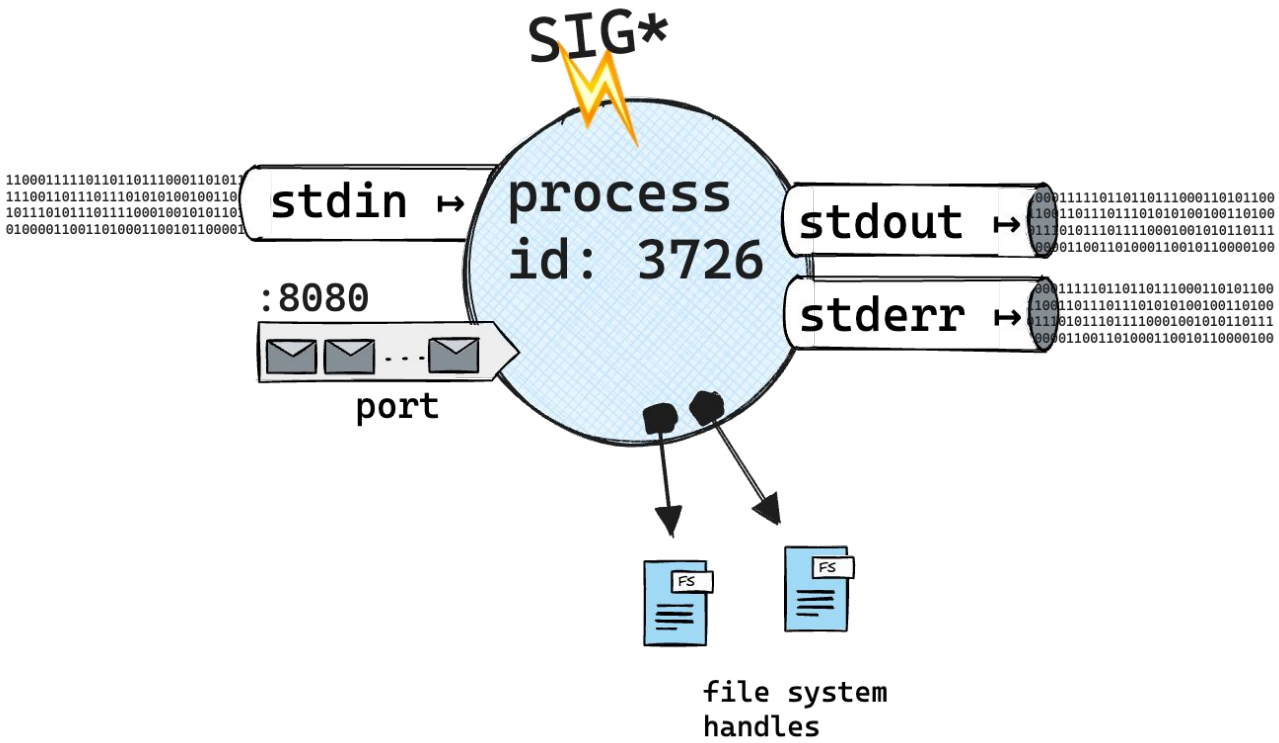


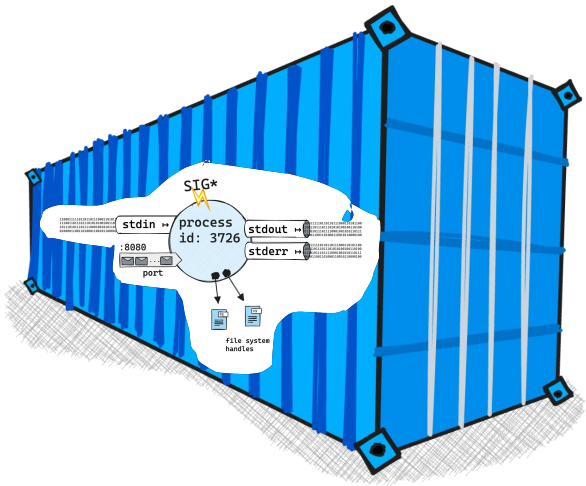
file system handles

Rough Model of a Process



Rough Model of a Process





New questions

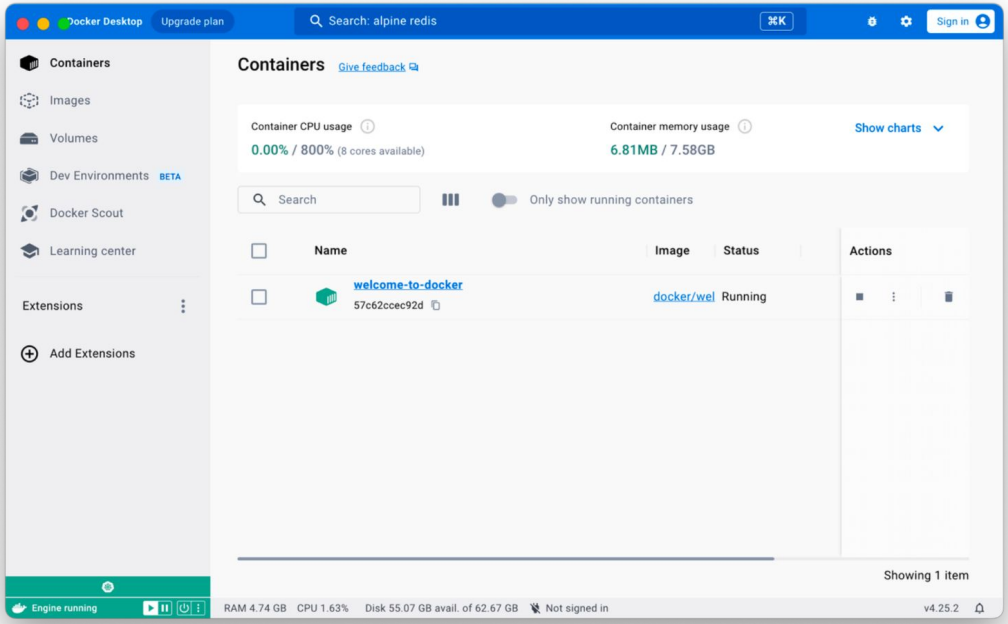
- How do we make a container?

✅What goes on inside?

- What does the "port" do with them?

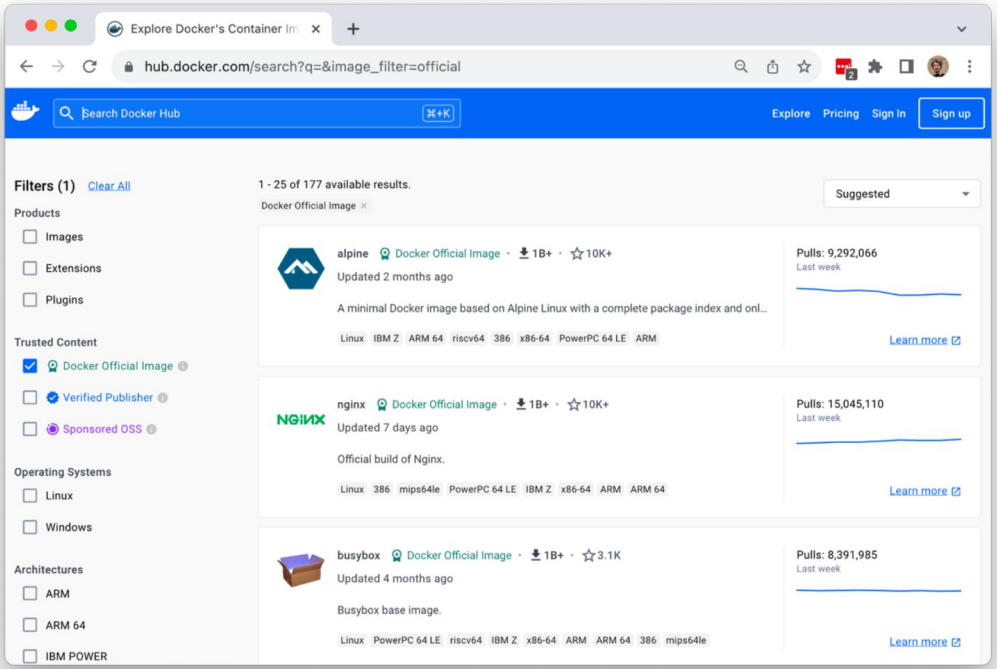
DEMO:

docker & Dockerfile



DEMO:

hub.docker.com

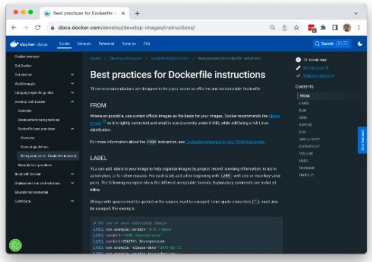


Docker Lab:

Get started with Docker apps in Visual Studio Code | Microsoft Learn

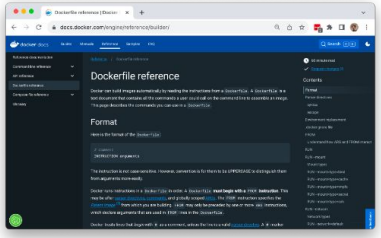
https://learn.microsoft.com/en-us/visualstudio/docker/tutorials/docker-tutorial

DOCS:



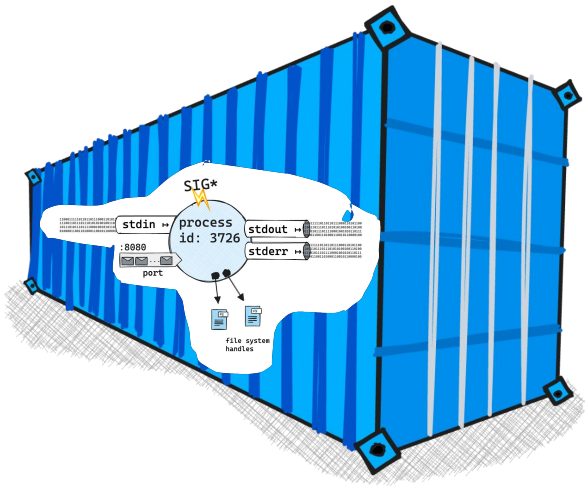
Best Practices

https://docs.docker.com/develop/develop-images/instructions/



Dockerfile Reference

https://docs.docker.com/engine/reference/builder/

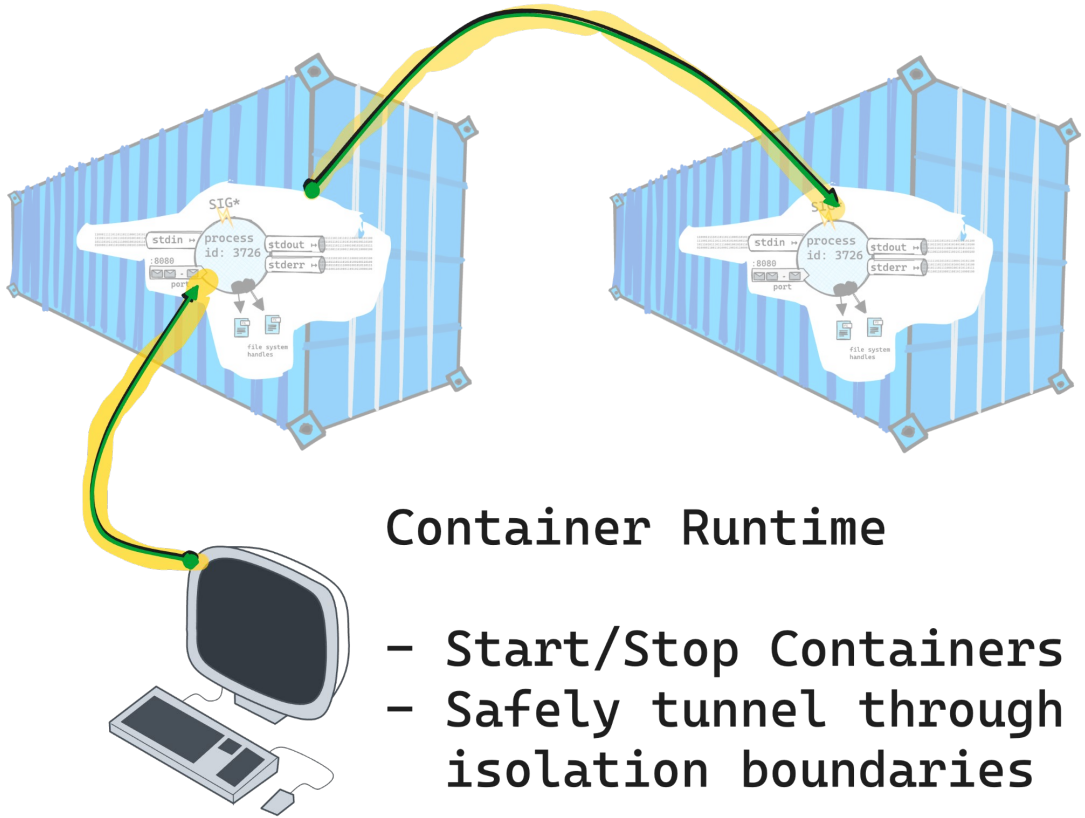


New questions

✅How do we make a container?

✅What goes on inside?

- What does the "port" do with them?

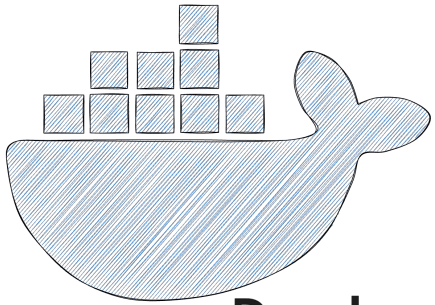


Container Runtime

- Start/Stop Containers
- Safely tunnel through isolation boundaries

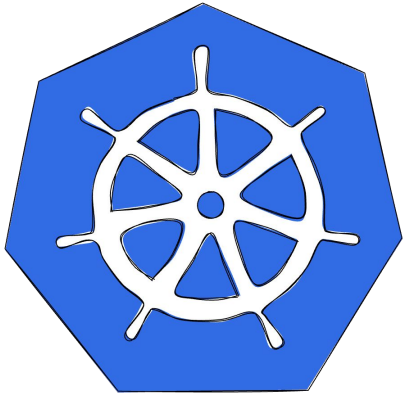
Lab 03

Kubernetes and Container Orchestration



Docker

- ⭐ convenient
- ⭐ builds images
- ⭐ machine-scale
- 😿 datacenter-scale
- 😿 fault-tolerant
- 😿 what is a deployment?

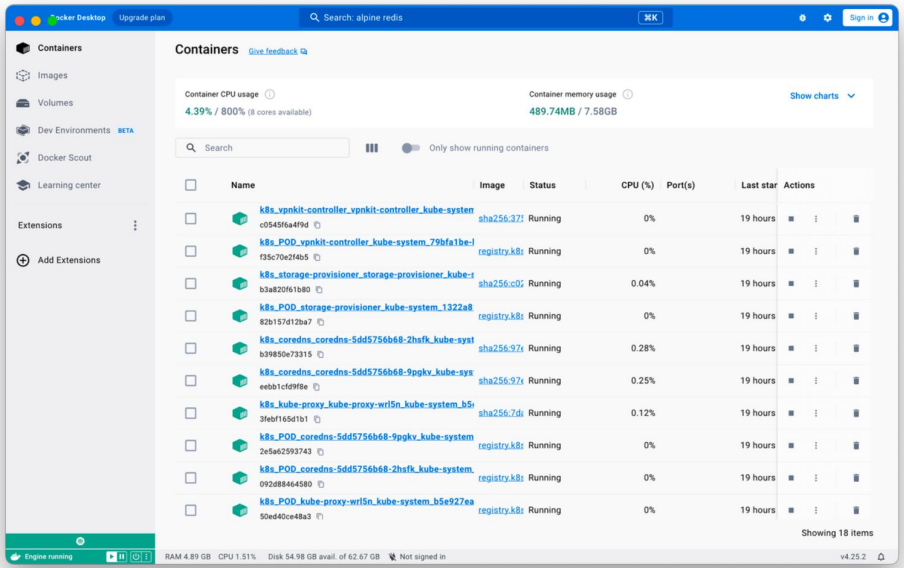


Kubernetes

- ⭐ datacenter-scale
- ⭐ fault-tolerant
- ⭐ manages deployment

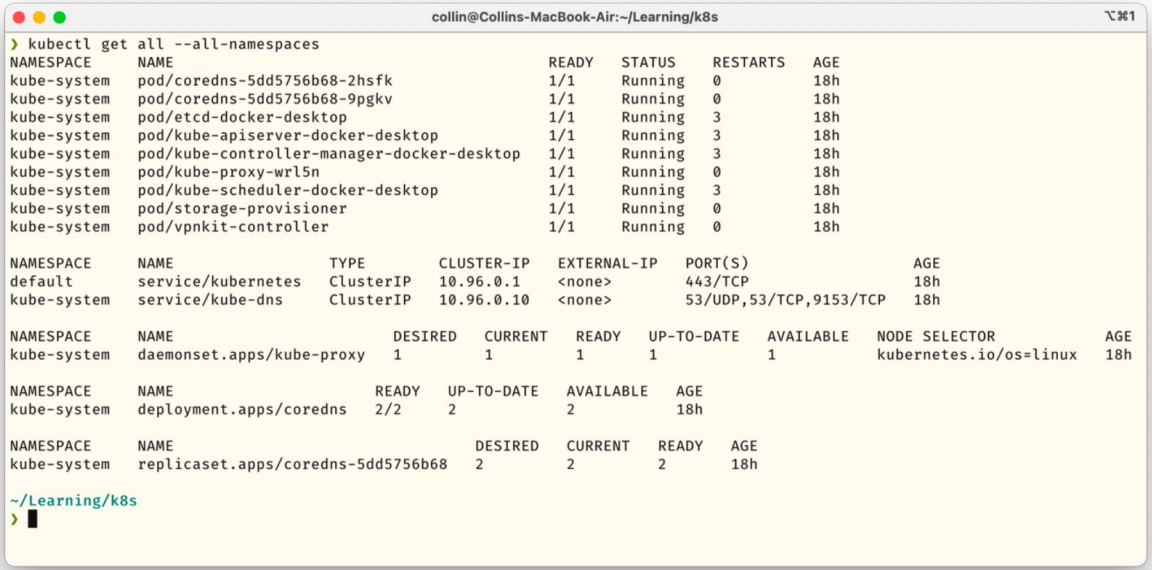
Install Kubernetes

DEMO:

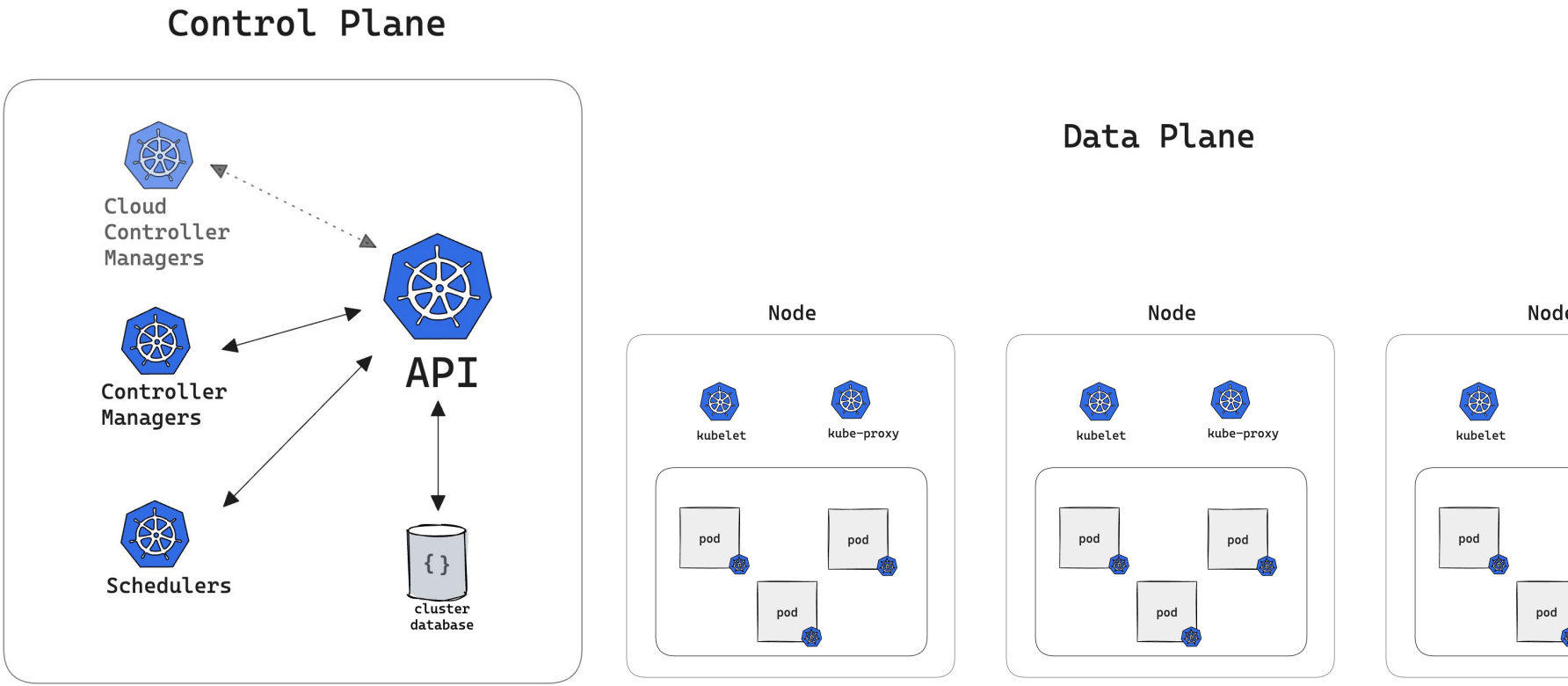


Explore kubectl

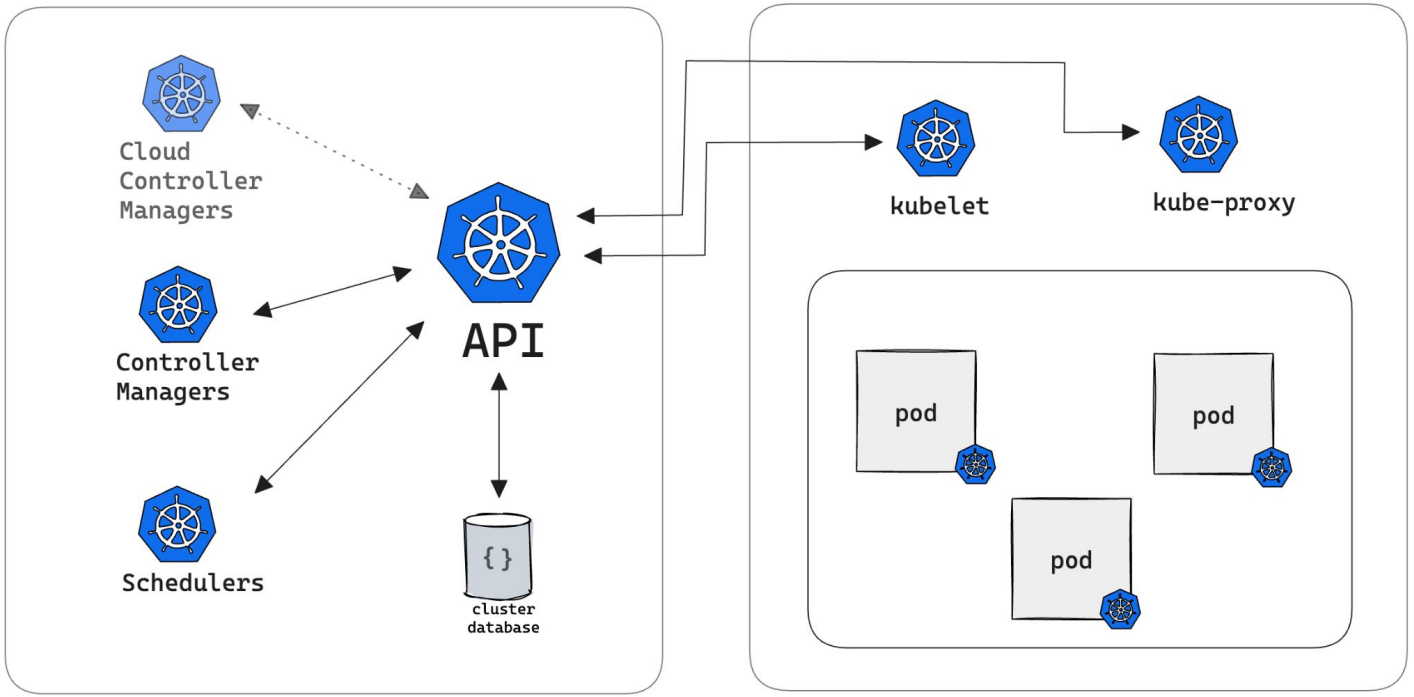
DEMO:



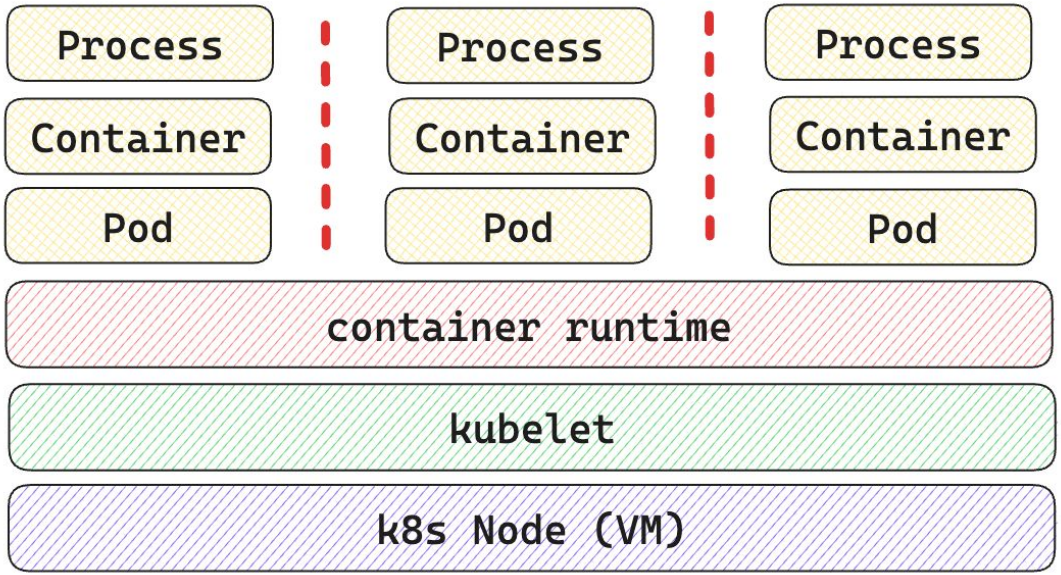
Architecture of k8s System



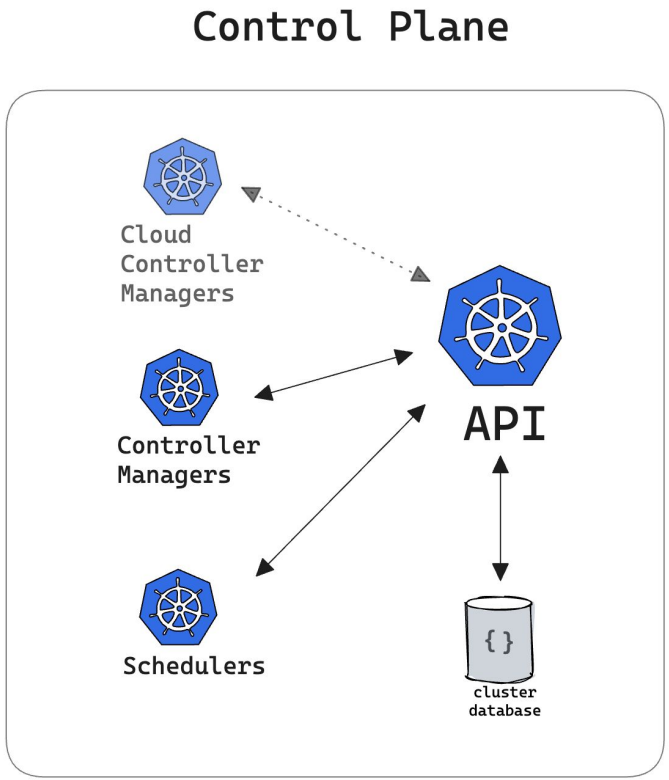
Architecture of k8s System



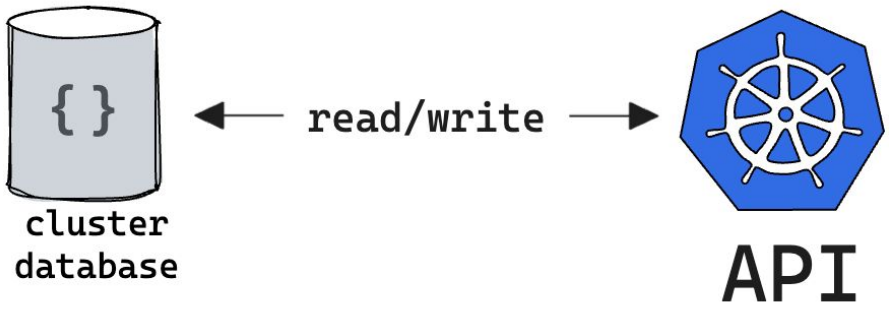
In kubernetes, our "process stack" looks a bit like this:



Architecture of k8s System

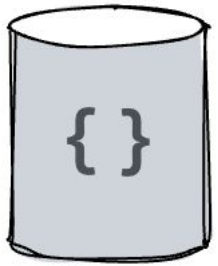


The heart of the kubernetes architecture



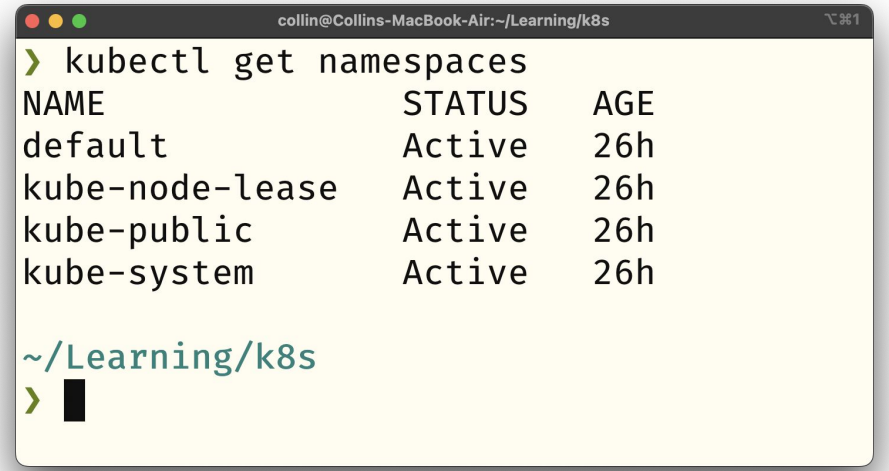
etcd

A distributed, reliable key-value store for the most critical data of a distributed system

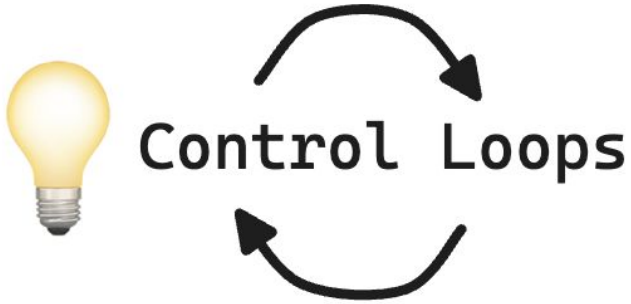


cluster database

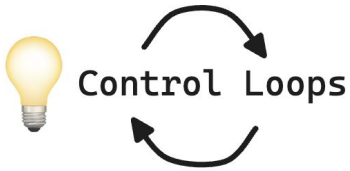
API



kubectl is sending API requests to this server



control loops are a
key concept k8s



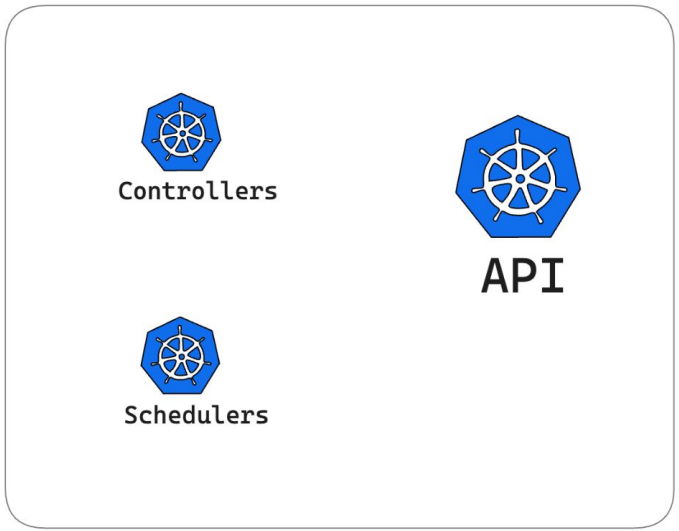
control loops are a
key concept k8s

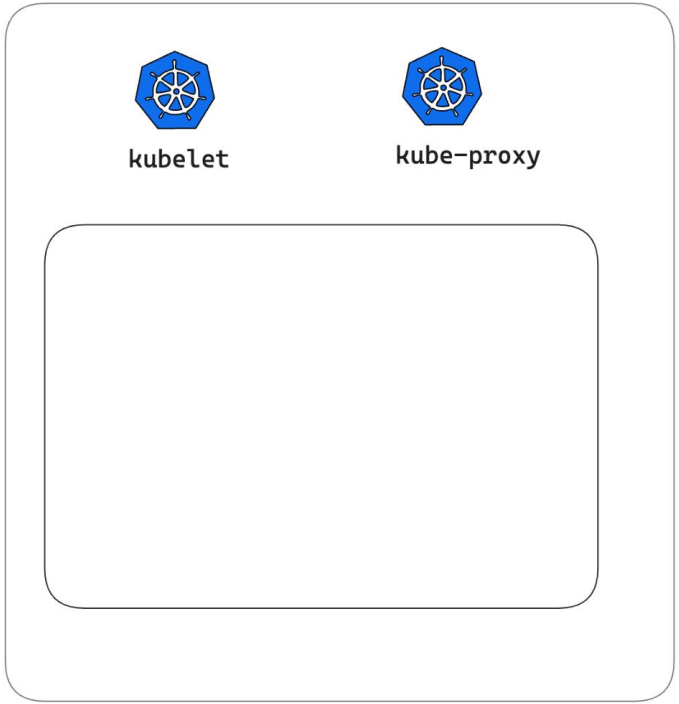
io.k8s.api.core.v1.Pod (v1@pod.json)

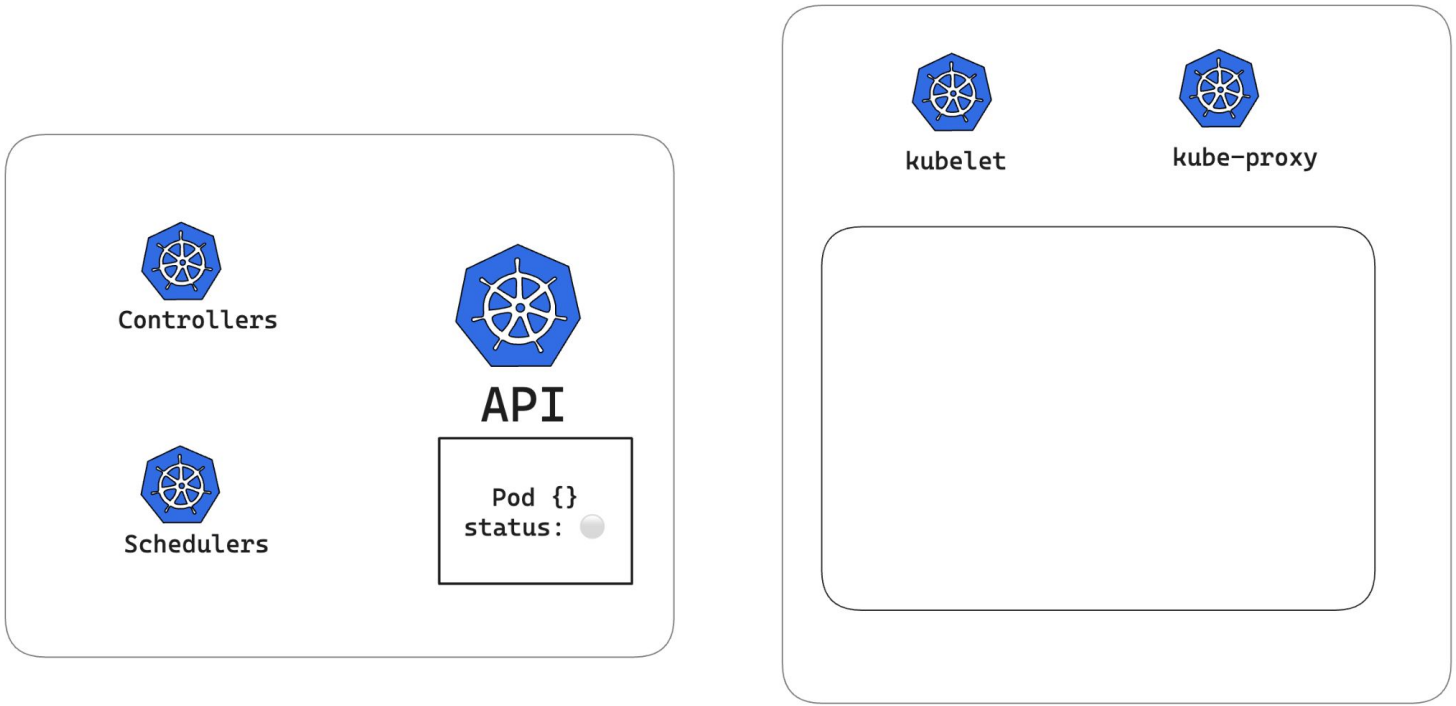
```yaml
apiVersion: v1
kind: Pod
metadata:
  name: hello-world
  labels:
    app: hello-world
spec:
  containers:
    - name: hello-world
      image: hello-world:v6
      ports:
        - containerPort: 80
      resources:
        limits:
          memory: "256Mi"
          cpu: "250m"
```

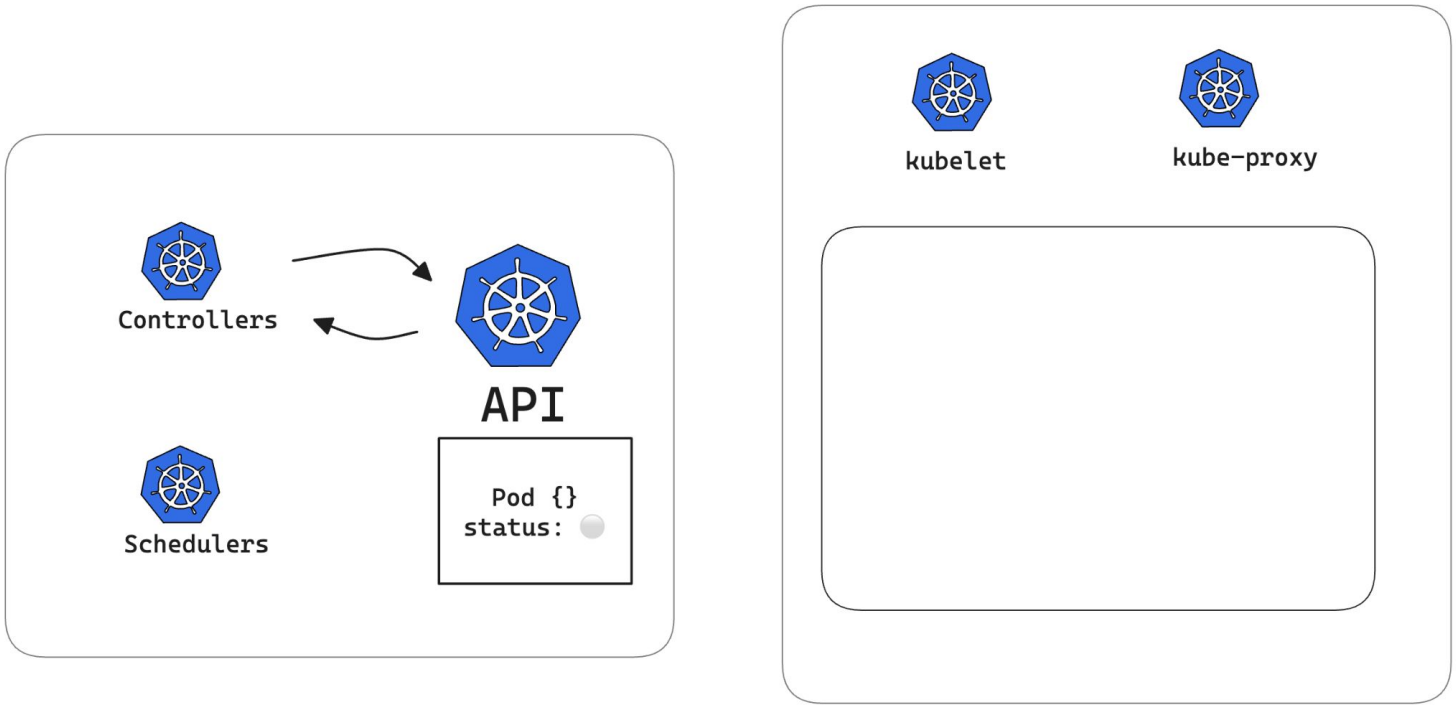
As are API Resources

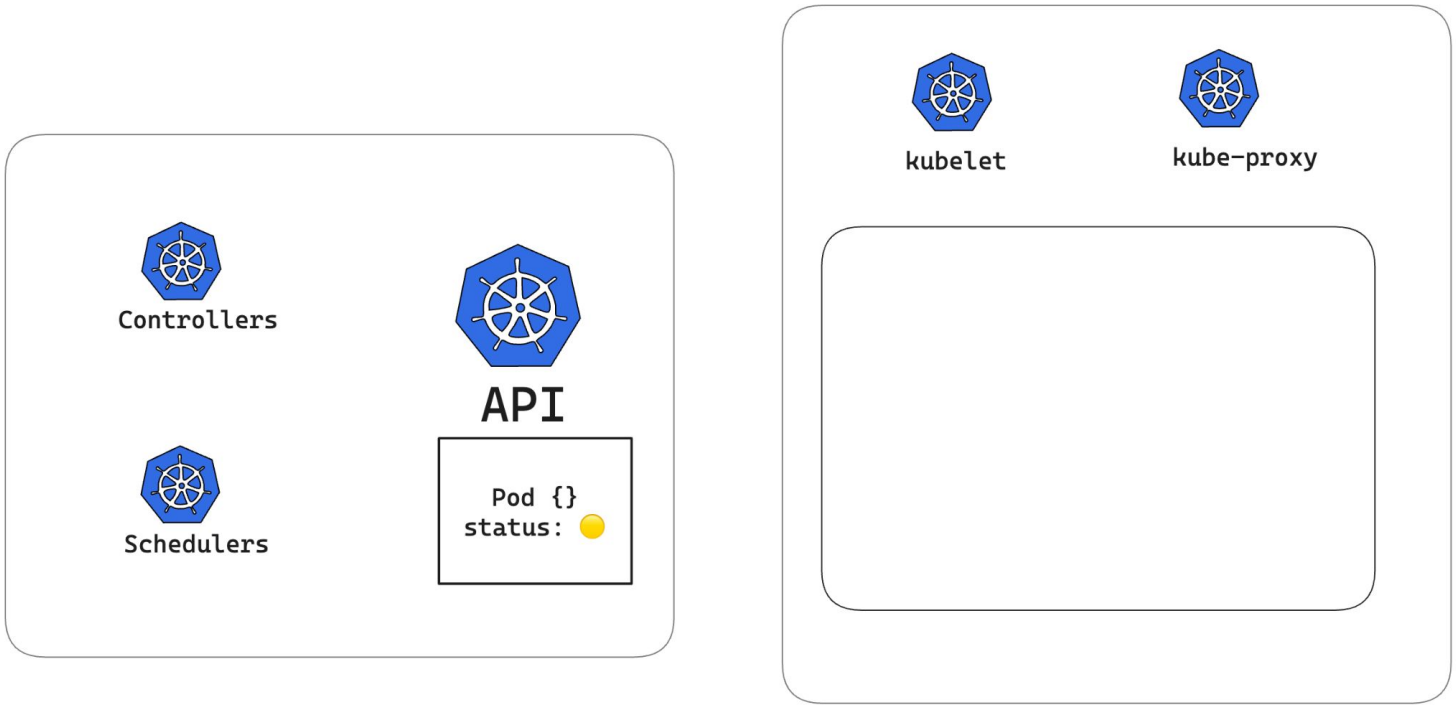
User: Create a Pod

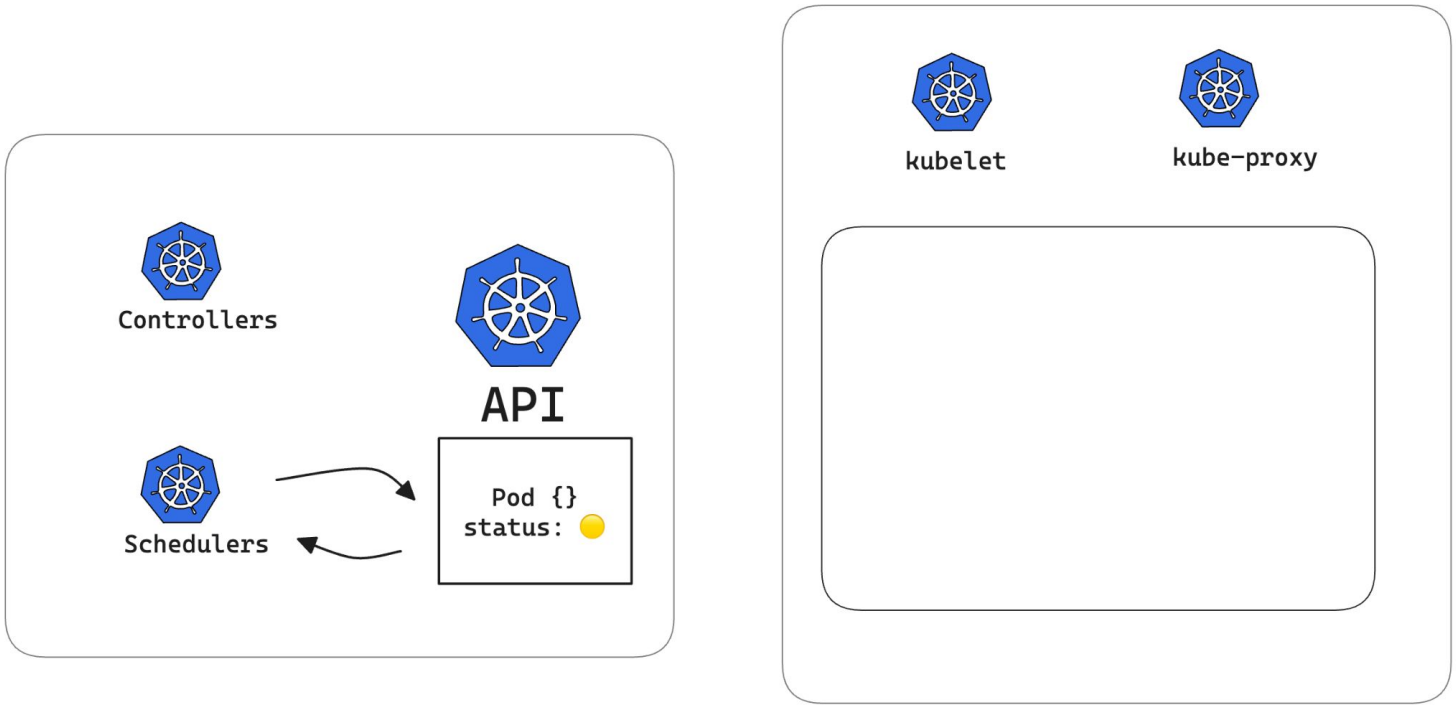


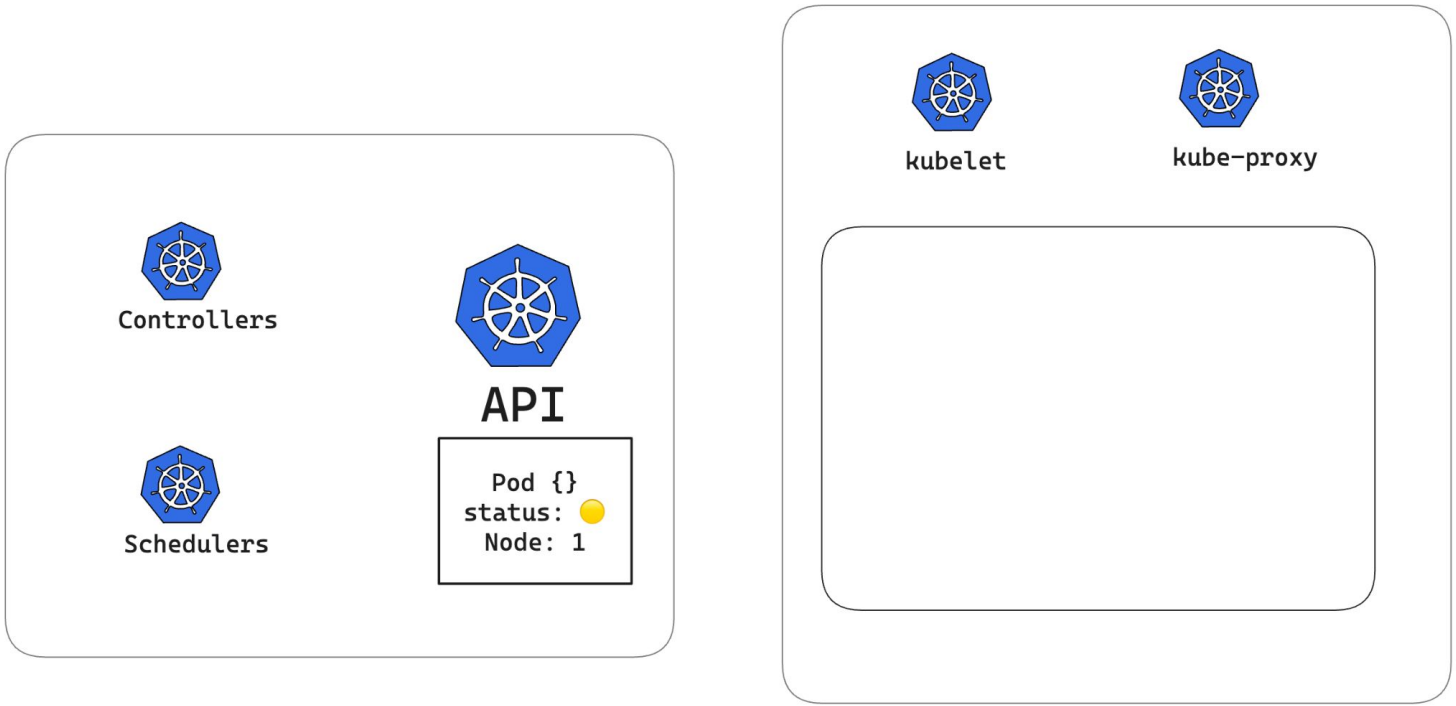


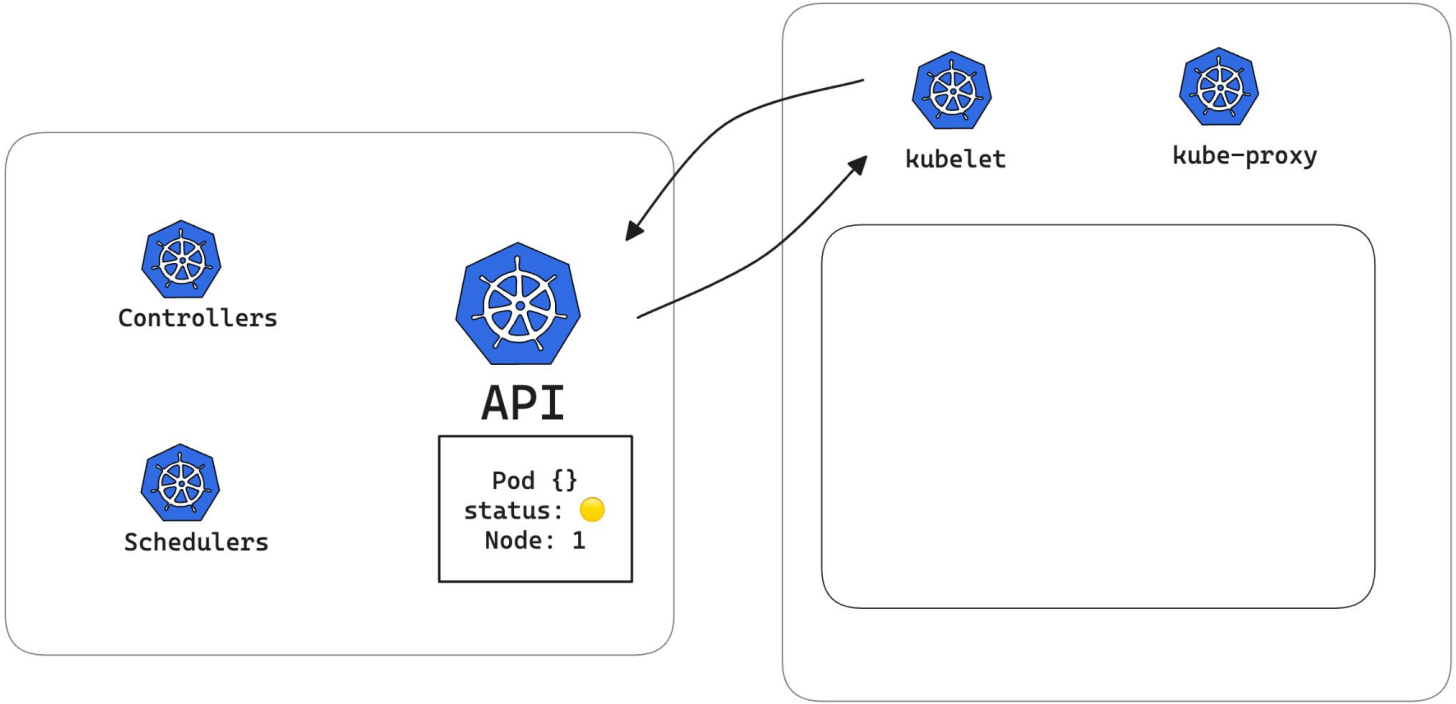

Controllers
API
Schedulers
Pod {}
status:
kubelet
kube-proxy

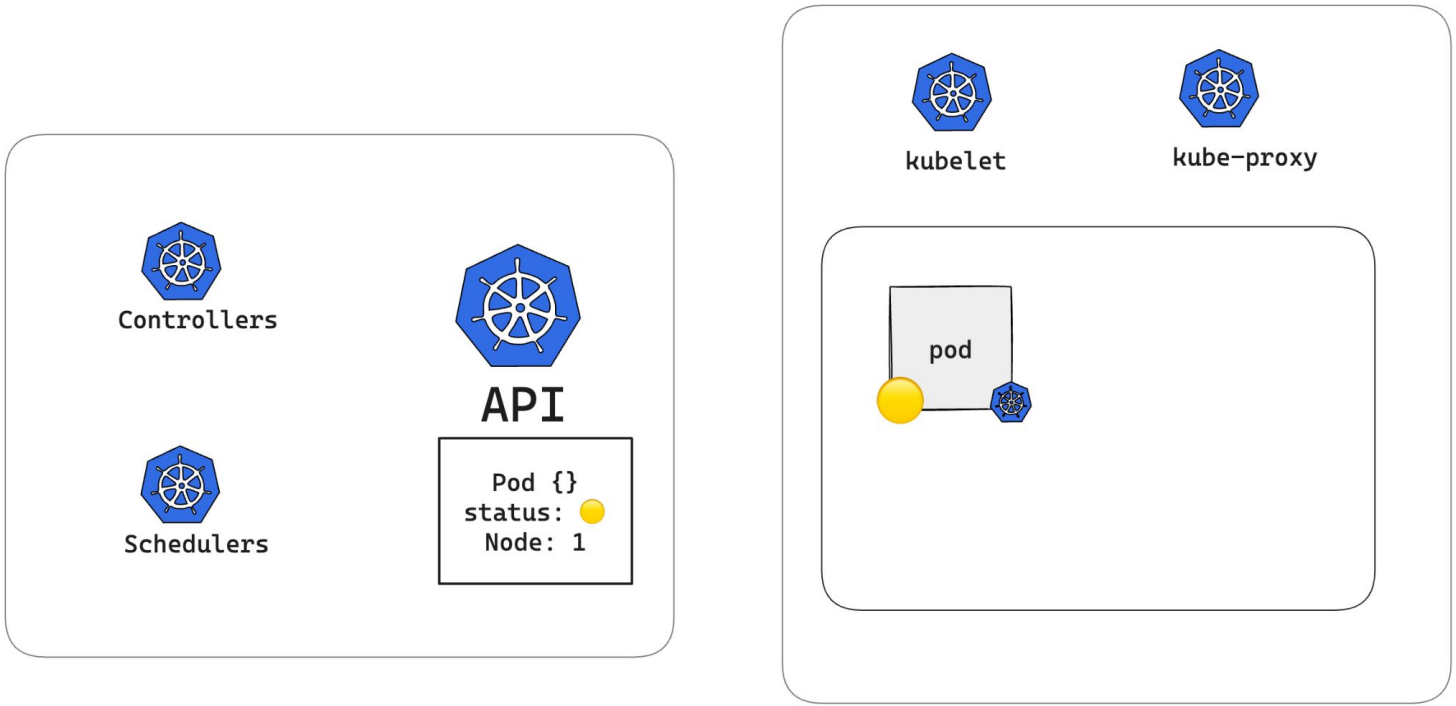

Controllers
API
Schedulers
Pod {}
status:
kubelet
kube-proxy

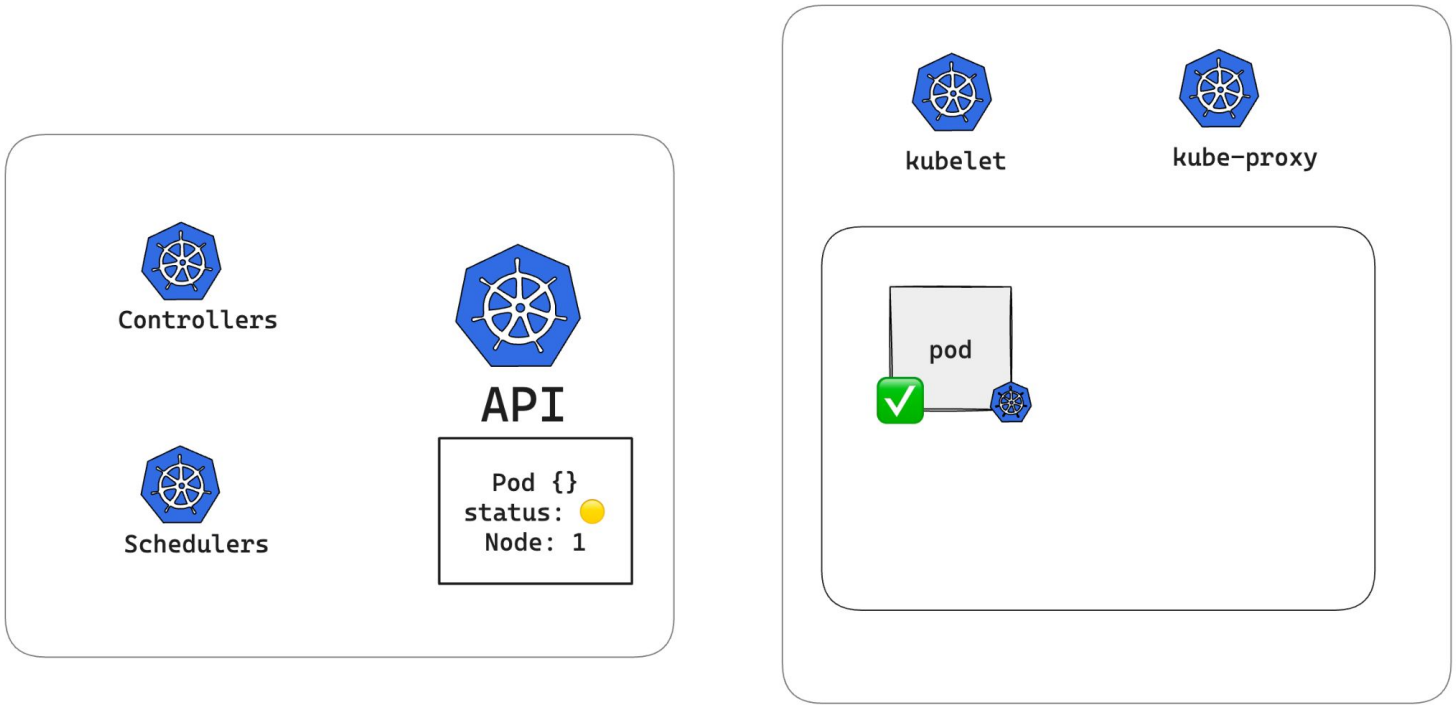

Controllers
API
Schedulers
Pod {}
status:
kubelet
kube-proxy

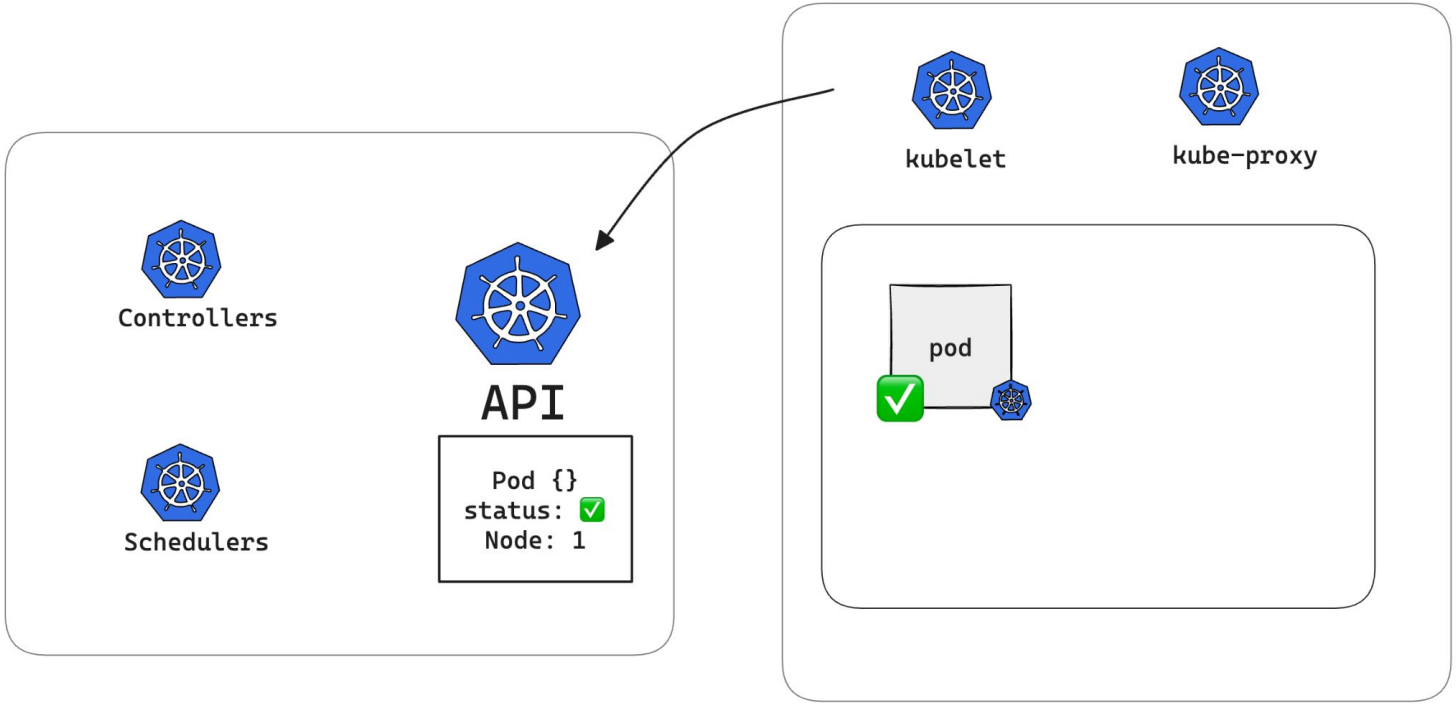

Controllers
API
Schedulers
Pod {}
status:
kubelet
kube-proxy

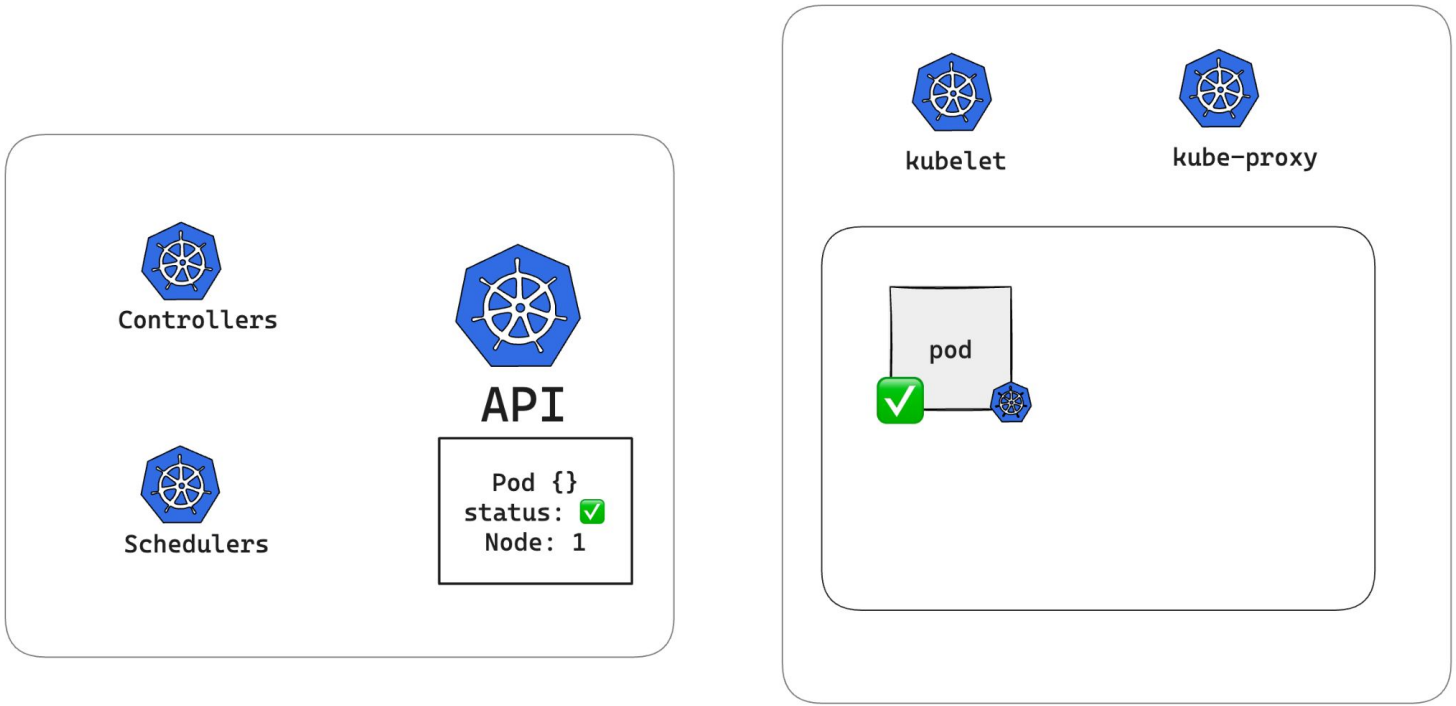

Controllers
API
Schedulers
Pod {}
status:
Node: 1
kubelet
kube-proxy

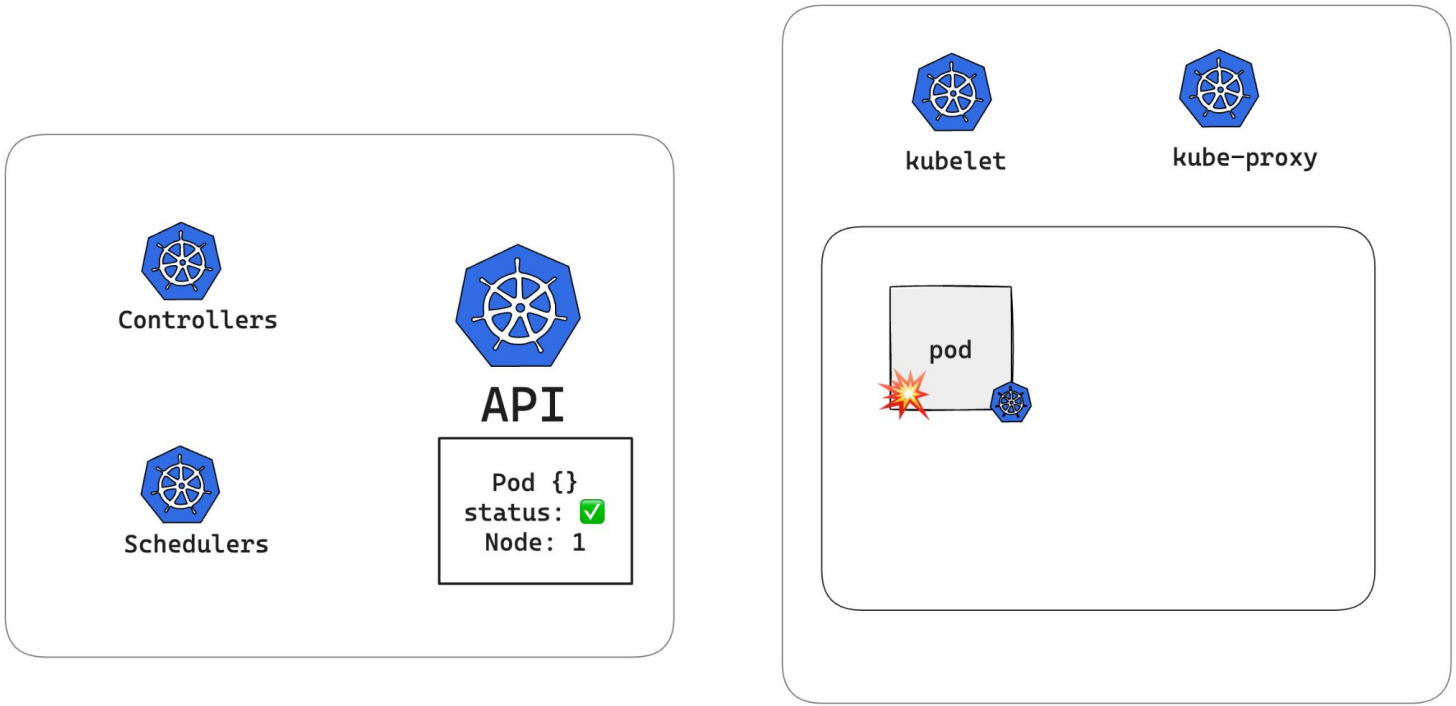

kubelet
kube-proxy
Controllers
API
Schedulers
Pod {}
status:
Node: 1

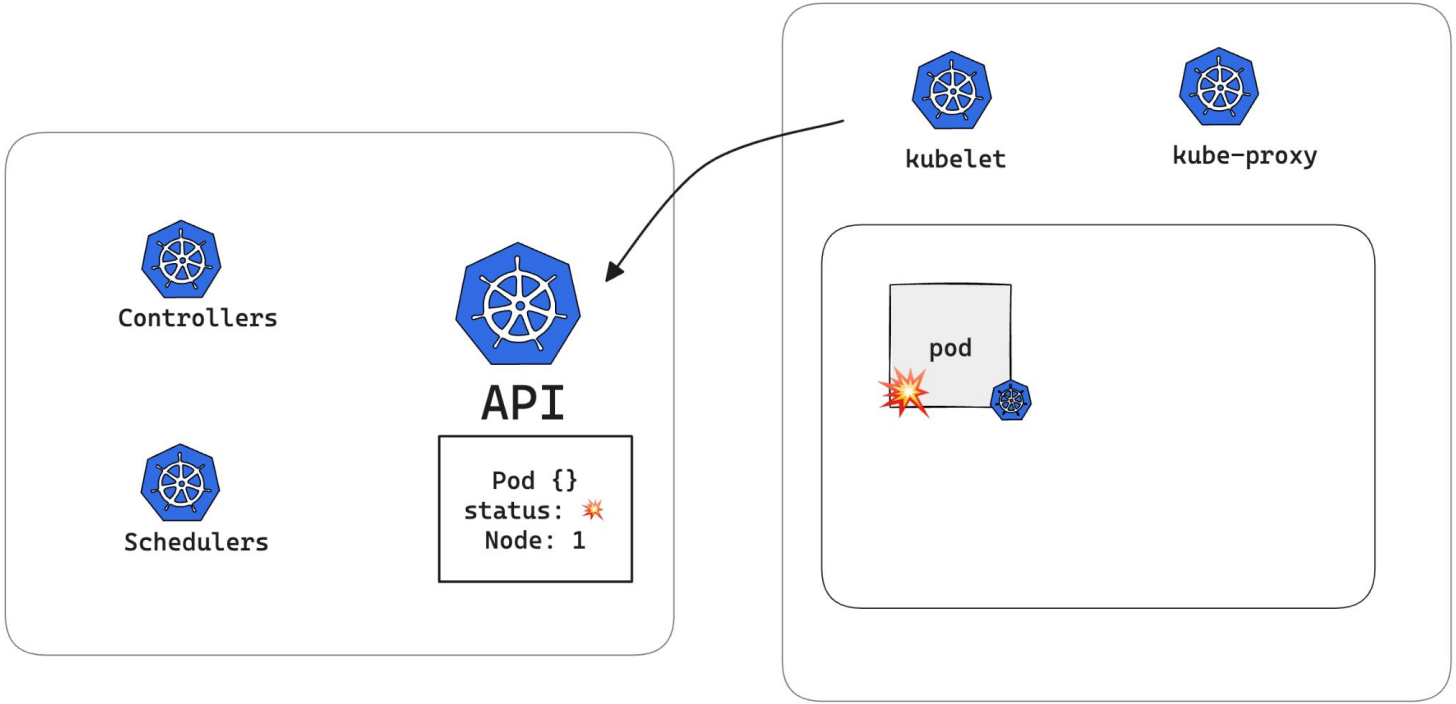

Controllers
API
Schedulers
Pod {}
status:
Node: 1
kubelet
kube-proxy
pod

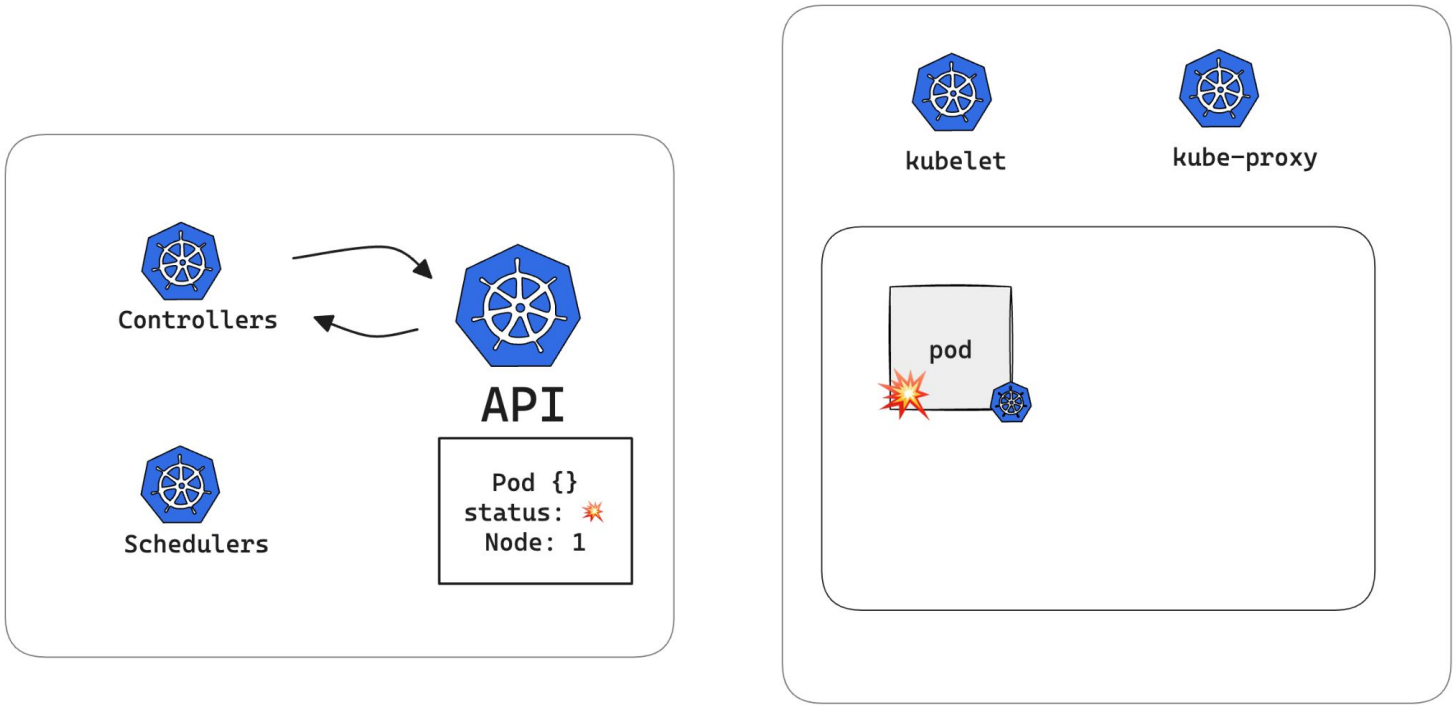

Controllers
API
Schedulers
Pod {}
status:
Node: 1
kubelet
kube-proxy
pod


kubelet
kube-proxy
Controllers
API
pod
Schedulers
Pod {}
status: ✅
Node: 1

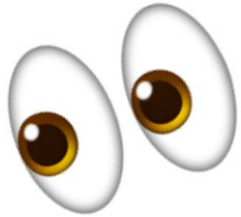

Controllers
API
Schedulers
Pod {}
status: ✅
Node: 1
kubelet
kube-proxy
pod


Controllers
API
Schedulers
Pod {}
status: ✅
Node: 1
kubelet
kube-proxy
pod


kubelet
kube-proxy
Controllers
API
pod
Schedulers
Pod {}
status:
Node: 1


Controllers
API
Schedulers
Pod {}
status:
Node: 1
kubelet
kube-proxy
pod

User: Create a Pod



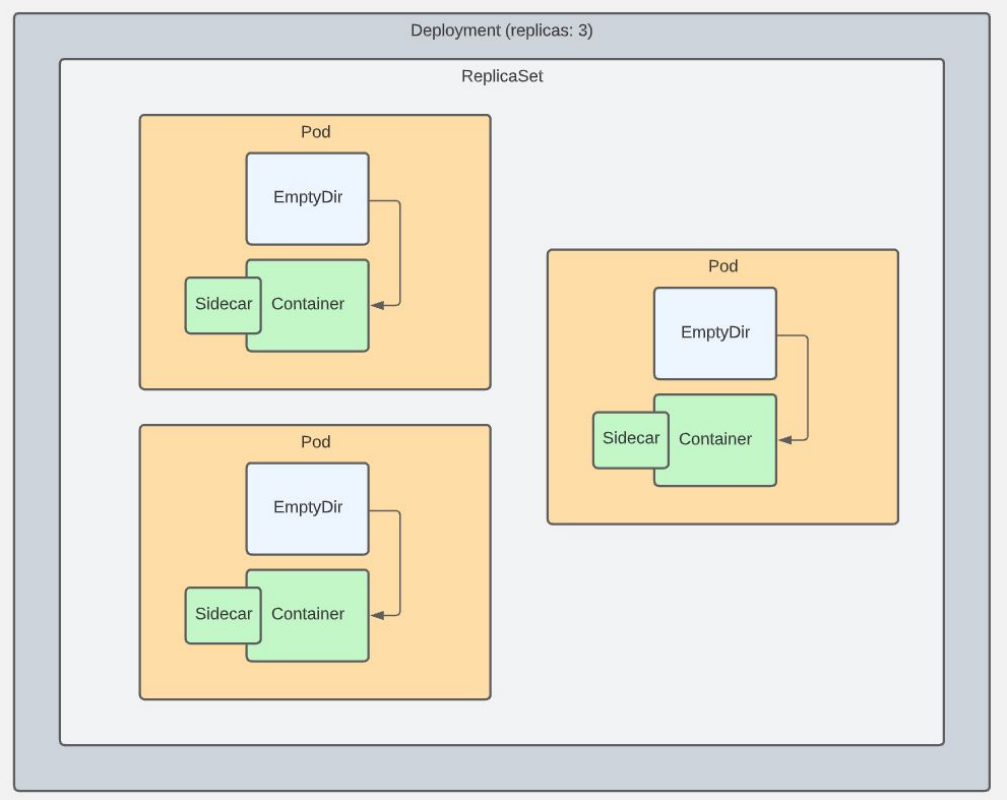
Deployment ✅

User: Create a ~~Pod~~

Deployments

- Every time the application code changes, a new version of the application container is built, and then there is a need to update the Deployment manifest with the new version and tell K8s to apply the changes
- K8s will then handle the rolling-out of this newer version, terminating pods with the old version as it spins up the new pods with the updated container
- This means that at some point we will have multiple versions of the same application running at the same time

Pods, Deployments, & ReplicaSets



Question for the Group

This component is used in a Kubernetes cluster to store state about the cluster and the resources it is responsible for:

1. API server
2. Scheduler
3. etcd
4. Controllers

Type your choice in the chat…

Question for the Group

This component is used in a Kubernetes cluster to monitor current state against desired state, and notify the cluster when an update is required to resolve any gaps:

1. API server
2. Scheduler
3. etcd
4. Controllers

Type your choice in the chat…

Question for the Group

This component provides the programmatic interface to the Kubernetes cluster and is used by other components and users for retrieving data about the cluster and applying changes to the cluster:

1. API server
2. Scheduler
3. etcd
4. Controllers

Type your choice in the chat…

Node Components: Container Runtime

- The software that runs the containers in the pod
- Docker is most common
- Others are containerd and CRI-O
- They need to comply with a standard called Kubernetes CRI

Addons: Cluster DNS

- A DNS system that runs in a pod in your cluster
- It also spins up a k8s Service
- Allows the use of consistent DNS names instead of IP addresses

k8s Probes

- The pods in a node are not always up and running, ready for incoming traffic
- We need a way to check
- The kubelet does this through periodic “liveness” checks into the pods in its node

kubectl

- kubectl (“kube-cuttle”) is command-line tool used to interact with API
- May require installation (automatic if using k8s via Docker Desktop)
- Config file defines details of connection to cluster (e.g., ~/.kube/config)
- Run `*kubectl cluster-info*` to confirm connectivity
- Run `*kubectl*` from command-line (with arguments) to see options
- NOTE: k8s in Docker Desktop should handle most of this for you

Lab 04

Pod Configuration

- Two types of pod: single container and multiple container
- Making a single container pod is simple – use kubectl run

$ kubectl run <name of pod> --image=<name of the image from registry>

Pod Design Patterns

- Usually, you will have one container per pod
- There are cases when multi-container pods make sense: When both pods have the same lifecycle, or they MUST run on the same node.
- An example: The primary container needs a helper process – it needs to be in the same node to be utilized.
- There are three main patterns for multi-container pods:
- Sidecar pattern
- Adapter pattern
- Ambassador pattern

Multi-container Pod: Sidecar Pattern

- Main App plus a “helper” app that isn’t part of the main app
- Example: Main app is a web app; helper is a logging utility

Multi-container Pod: Adapter Pattern

- Used when you need to standardize output form various apps in the cluster
- Each app type may format output in a unique way, but the cluster's monitoring needs to work with standardized formats
- Adding an adapter container to the pod for each app lets you still use the normal app output format in other areas, but have the cluster-level data work use a standardized format

Multi-container Pod: Ambassador Pattern

- One more way to allow the pod's containers to communicate with the outside world
- Add an "ambassador" container to the pod that acts as a proxy for network traffic to and from the primary container
- This lets you control traffic patterns based on the specific use case of the pod in question.
- Example: database connection in dev/qa/prod environments

Lab 05

Diving Deeper into Kubernetes

Namespaces in k8s

- Provide a grouping mechanism in Kubernetes
- Every k8s object belongs to a namespace
- Every Cluster automatically includes a default namespace

Namespaces in k8s

- Allow you create logical separation within a physical Cluster
- Provide a boundary for security and resource control
- Can explicitly create a namespace and deploy resources to it using namespace field in manifest or --namespace arg on kubectl

Namespaces in k8s

- Objects within a namespace are isolated from others
- Enables creation of multiple instances of same apps with same names
- Resources from one namespace can communicate with another namespace using Services
- Controller only looks for matching resources in its namespace

Contexts

- Can be used to define connection details for a k8s Cluster
- Sets the default namespace to be used
- Prevents need to specify namespace – will use context to automatically set if excluded (like with default in standard install)
- Able to switch between contexts using use-context command

Contexts

- Can also be used to manage switching between different Clusters (local or remote)
- For remote, API server will be secured with TLS and kubectl uses a client cert for AuthN

ConfigMaps vs. Environment Variables

- A ConfigMap is an API object that lets you store configuration for other objects to use
- These are key/value pairs
- This lets you decouple environment-specific config data from your container images, making your apps more portable
- CAUTION: ConfigMaps do not provide encryption or secrecy. Use a Kubernetes Secret for that.

ConfigMaps

- Can be:
  - Set of key-value pairs
  - Blurb of text
  - Binary files
- One pod can use many ConfigMaps and one ConfigMap can be used by many pods
- Data is read-only – pod can’t alter

ConfigMaps as Env Vars

- Can be created from a literal:
  `*kubectl create configmap <name> --from-literal=<key>=<value>*`
- Can be created from a file (to group together multiple settings):
  `*kubectl create configmap <name> --from-env-file=<file-path>*`

Where <file-path> contains .env file like:

key1=value1
key2=value2

- Can be created from a .yml definition file (like all things k8s)

ConfigMaps

- Can be presented as files inside directories in the container
- Uses volumes – making contents of ConfigMap available to pod
- Uses volume mounts – loads contents of ConfigMap volume into specific container path in pod
- Can contain settings to override defaults

ConfigMaps - Precedence

- If env vars defined in multiple places, definitions in `env` section in pod spec will trump others (localized)
- Typical approach:
  - Default app settings “baked in” to container image (e.g., to support dev mode)
  - Specific settings for an environment stored in ConfigMap and surfaced to container filesystem
  - Merges with default settings (overwriting where applicable)
  - Final tweaks can be accomplished with env variables in pod spec

Secrets in Pods

- Secrets let you store and manage sensitive information, such as passwords, OAuth tokens, and ssh keys.
- A pod needs to reference a secret. There are three ways get to the secret:
- As a file in a Volume that is mounted to a container;
- As an environment variable for a container;
- By the kubelet, when it pulls images for the pod
- NOTE: The default config for a Secret does NOT have encryption – data is plain text. You need to configure “Encryption at Rest” for the Secret”, and Role Based Access Control in your cluster

Lab 06

Lab 07

k8s Services

- Services are an abstraction
- They define a set of pods, and an access policy for them
- A pod has an IP address
- Pods are ephemeral – what happens to IP traffic if they die?
- Pods in a service can have fixed IP addresses that stay the same even if the actual pod dies and is spun up again
- Services are k8s objects; they are created like any other

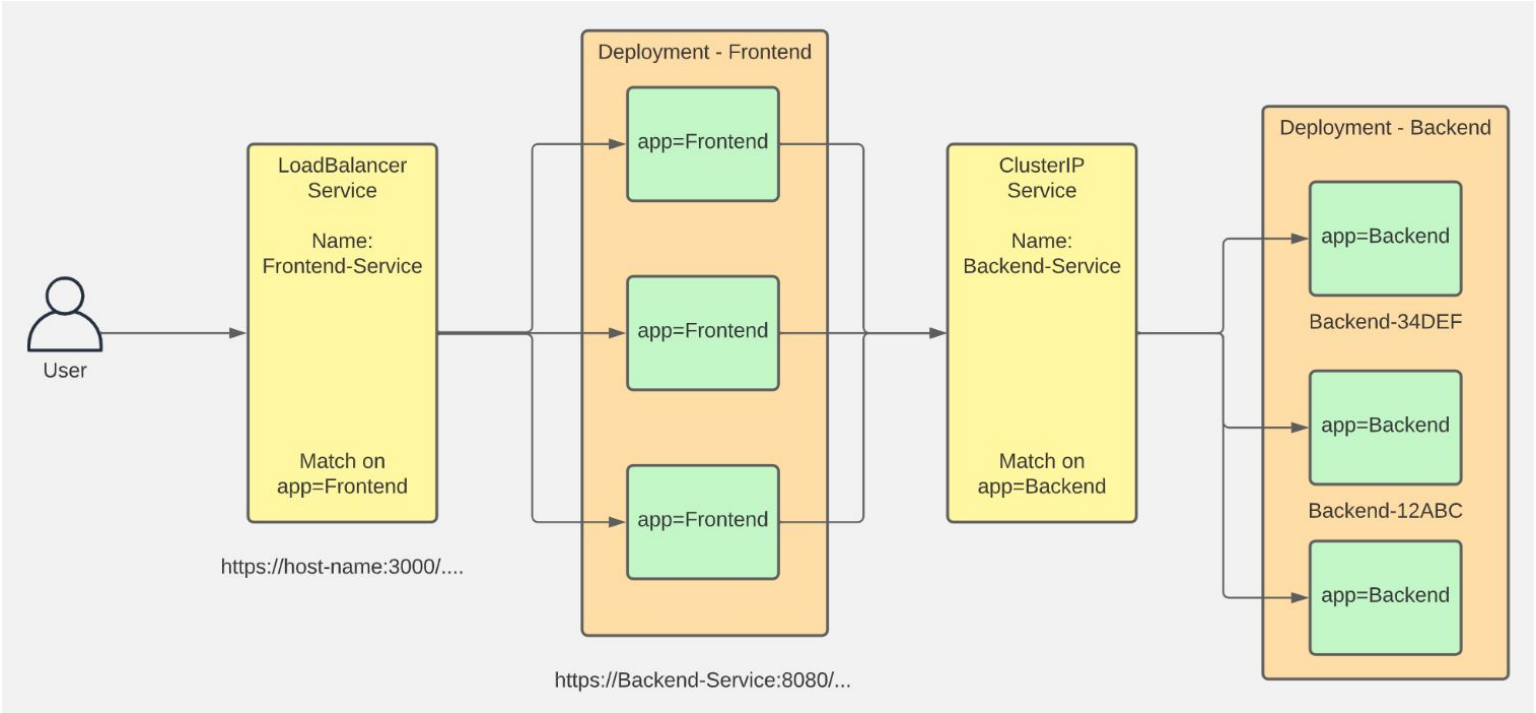
k8s Service Types

- ClusterIP (default): IP is internal to the cluster
- NodePort: IP for the port is exposed with a static IP address
- LoadBalancer: IP traffic to nodes is managed by external load balancer
- ExternalName: IP traffic is routed by DNS Name (foo.example.com)

Cluster Access – Internal vs. External

- You can access the cluster API using kubectl (“proxy”)
- You can get libraries for popular languages that can access the cluster
- k8s objects (pods) can also access the API internally
- This uses an internal DNS
- “kubernetes.default.svc” maps to the API server
- The pod usually uses a “sidecar” container to do this

Services


Deployment - Frontend
app=Frontend
app=Frontend
app=Frontend
LoadBalancer
Service
Name:
Frontend-Service
Match on
app=Frontend
User
https://host-name:3000/....
ClusterIP
Service
Name:
Backend-Service
Match on
app=Backend
Deployment - Backend
app=Backend
Backend-34DEF
app=Backend
Backend-12ABC
app=Backend
https://Backend-Service:8080/...

Lab 08

Lab 09

Lab 10

Container File System

- Each container in a pod has its own filesystem built by k8s
- Can be built from multiple sources, including:
  - Layers from the container image
  - Writable layer for the container
- Can be expanded with other sources like ConfigMaps and Secrets – mounted to a specific path in a container
- Volumes can provide another source of storage – volumes are defined and mounted to containers

EmptyDir

- Empty directory stored at pod level vs. container level
- Mounted as a volume into the container so visible there
- Any data stored there from the container remains in pod between restarts
- Replacement containers can access data from predecessors

HostPath

- Data physically stored on node in cluster
- Extends beyond lifecycle of pod
- Available to replacement pods (but only if run on same node)
- Mounted as a volume into the container like others
- However, can have issues:
  - In cluster with multiple nodes, limits usefulness
  - If not careful, can expose large blocks of host data

PersistentVolume

- Like pods (abstraction over computer) and services (abstraction over network), persistent volumes provide abstraction over storage
- k8s object that defines available section of storage
- Can be “mapped” to shared storage (like NFS) or locally (for dev/test purposes)

PersistentVolumeClaim

- Pods are not allowed to use persistent volumes directly
- Instead, pods claim using PersistentVolumeClaim object type in k8s
- Requests storage for a pod
- k8s handles matching up a claim to a persistent volume
- Provides virtual storage abstracted from actual storage

Dynamic Provisioning

- Static provisioning accomplished by explicitly creating persistent volume and then claim (k8s binds claim to volume)
- Dynamic provisioning supported as well
- You create the persistent volume claim and let persistent volume be created on demand by cluster
- Can leverage storage classes for defining and matching types of storage

Storage Classes

Defined by three properties:

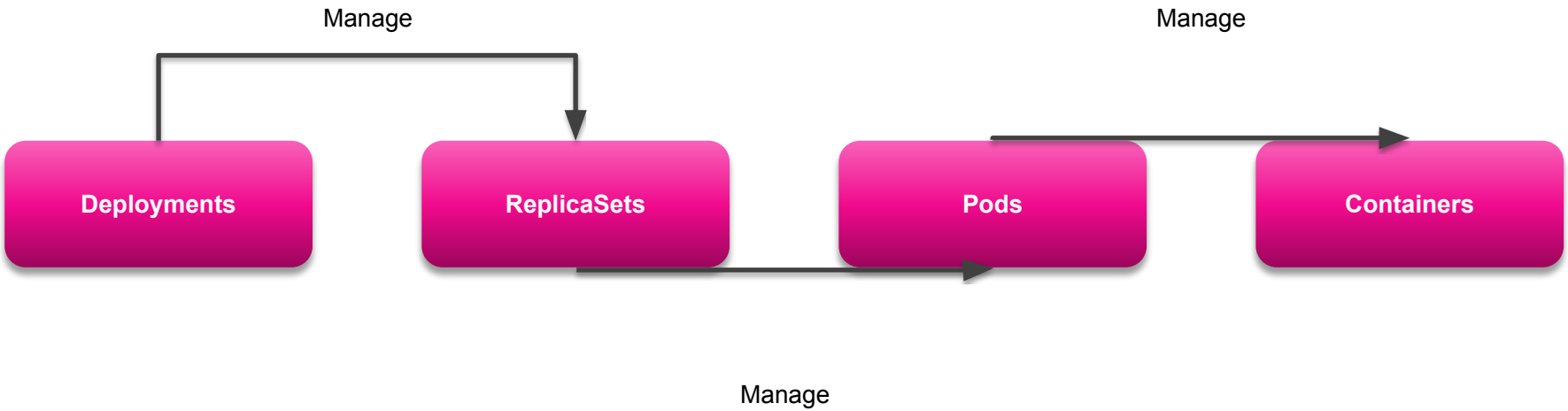
- provisioner – component that creates persistent volumes on demand
- reclaimPolicy – what to do with dynamically created volumes when claim is deleted
- volumeBindingMode – eager vs. lazy binding/creation (at same time as claim creation or only when pod using the claim get created)

Also, able to create custom storage classes defined by same properties

Lab 11

Demo – Deploying WordPress and MySQL

Scaling k8s


Manage
Manage
Deployments
ReplicaSets
Pods
Containers
Manage

ReplicaSets

- Can be created directly as an object in k8s
- However, better practice would be to create/manage through deployments
- `replicas` property in pod spec in deployment can be used to build out multiple replicas for a pod config
- If not included, defaults to 1

Setting Limits in Deployments

- When defining pod spec in deployment, can use limits to manage
- `resources` attribute defines compute resource requirements for pod
- `limits` can be used to apply limits (e.g., in amount of memory and cpu)
- Usually better to optimize once running vs. prematurely optimizing
- Target environment when running will help determine correct settings

DaemonSets

- daemon is usually a system process that runs constantly as single instance
- In k8s, DaemonSet runs single replica of a specific pod on every node in cluster
- Usually used for infrastructure-level concerns:
  - Logging/central data collection
  - High availability for a reverse proxy running in cluster
- Looks similar to other controllers (e.g., Deployment) but no replica count

Using DaemonSets in Deployments

- Control loop watches for nodes joining cluster
- New pod instance for the daemon will be started on new node
- Works to stop/remove daemon pod(s) if node(s) taken out of cluster

Using DaemonSets in Deployments

- Can also target just a subset of nodes for daemon pod
- As with most other things in k8s, use label matching
- Watches for nodes join cluster and for metadata matching label
- DaemonSet control loop is different from ReplicaSets because has to monitor combo of node activity and pod count

Lab 12

StatefulSets

- Represents a Pod controller that enables running apps at scale in a predictable manner
- Deployments/ReplicaSets create Pods with random names
- Also, starts Pods in parallel
- Some use cases (e.g., clustering) require a more deterministic approach

StatefulSets

- For example, creating a cluster of PostgreSQL instances
- With StatefulSets, can create Pods with predictable names
- Can access the Pods individually over DNS
- k8s will start the Pods in sequence

StatefulSets

- With a PostgreSQL cluster, able to earmark a Pod as primary
- Other Pods will wait for the primary and can use primary for replication
- Allows clustering of multiple database instances with read-replicas
- Pod name will be StatefulSet name followed by an index (starts at 0)

StatefulSets

- Provides stability for the deployment
- Replacement of missing Pods will respect this ordering
- If Pod 0 goes down, a replacement Pod 0 will be created (with same config)
- Other Pod indexes will stay in place

StatefulSets

- To get access to DNS names for individual Pods, create a “headless” service
- Create the service with a clusterIP of None
- Does not use fixed IP for the Service
- DNS entry for the service returns an IP address for each Pod in the StatefulSet
- Each Pod gets its own DNS entry as well

StatefulSets

- Can require adjustment to data storage approach
- With something like PostgreSQL, each database instance will write data in its own writable layer instead of a shared location
- Cannot simply mount a standard PVC for data storage as every Pod would try to write to that volume shared across all Pods

StatefulSets

- StatefulSet specification provides a volumeClaimTemplates field
- Can include storage class, capacity, and access mode definitions
- Each Pod in the spec gets its own dynamically-created PVC
- As previously discussed, creates PersistentVolumes for each using defined storage class (or default in not specified)
- This link between Pod and PVC remains in place even across Pod replacement

StatefulSets

- Keep in mind – StatefulSets are intended to provide a stable and predictable deployment environment
- Less flexible than other controller types
- Can make changes more impactful
- Be sure app requirements fit model

Jobs

- Provide options for running maintenance tasks in your k8s Cluster
- For example, with PostgreSQL cluster from previous, can schedule tasks against the DB “out-of-band”
- Tasks like backing up databases, data migration, data loads, etc.
- Defined with a Pod spec and run to completion as a batch process

Jobs

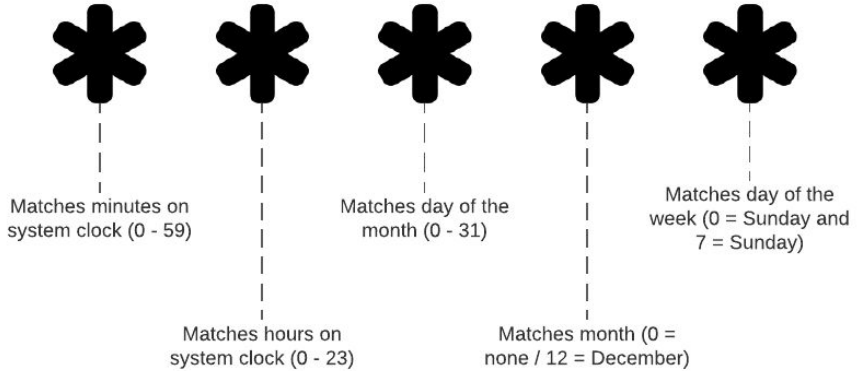
- Can run Pods for any container image but should represent a process with a discrete end (intended to run to completion)
- Jobs add labels to the underlying Pods created
- Includes a completions property for specifying how many times the Job should run

Jobs

- And a parallelism property for specifying how many Pods to run in parallel
- Enables distribution of Job processing for multiple types/instances across the Cluster
- Also, can assist with managing the speed of Job execution

CronJobs

- Specific type of Job for executing regularly and on a schedule
- Allows definition of a Job spec for use by CronJob for Job creation on the defined schedule
- Use the Linux Cron format



CronJobs

- Can move CronJobs to suspended state
- Logic needs to account for standard Pod lifecycle management
- Could include restarts or parallel runs – logic needs to support

Lab 13

Lab 14

Startup Probes

- First probe executed on startup and determines if app is initialized and ready for traffic
- Only runs on initialization – once complete, will not run again in the Pod
- Can specify number of probe attempts (failureThreshold)
- And number of seconds to wait between retries (periodSeconds)
- If all defined attempts fail, k8s will kill the Pod and startup a replacement

Readiness Probes

- k8s knows if Pod container is running but not if application inside is healthy (specific to the app)
- Defined in the Pod spec
- Execute health checks on a fixed schedule

Readiness Probes

- Act at the network level and manage routing for listening components
- If deemed unhealthy, Pod will be taken out of ready state and removed from list of active Pods
- k8s keeps running probe looking for Pods that have recovered for reassociation to the Service
- Different types of probes are supported – e.g., HTTP GET for web apps

Liveness Probes

- Use same health checks as readiness probes
- Action on failure is different
- Act at compute level and restart unhealthy Pods
- Different types of probes are supported – e.g., HTTP GET for web apps

Lab 15

Managing Logs

- k8s collects logs from container runtime and stores as files on node
- Goal is to get logs out of the container and potentially centralize for aggregated processing
- Often uses a sidecar container for the aggregation

Managing Logs

- Logs can benefit from normalization
- Parsing can be applied as the information passes through the pipeline
- Several 3rd party tools (e.g., Fluentd for log collection, Elasticsearch for log storage, and Kibana for view)

Deploying and Scaling Clusters

- Clusters can scale as needed
  - Increased load
  - Rolling cluster updates
- This is often managed by the cloud provider with cloud native

Managing Cluster Resources

- Container limits provide options for specifying limits at the container level
- Allow specification of memory and cpu limits to prevent a container from exhausting cluster resources
- Can be defined using “requests” and “limits” in container spec
- “requests” define what is requested (like a minimum) – container can use more than what’s defined if node provides
- “limits” are hard stops – if exceeded container will be replaced

ResourceQuota

- Also, able to apply resource limits at the namespace level
- Pod specs need to include a resource section
- Container limits are reactive; ResourceQuotas are proactive
- Pods won’t be created if they would exceed limit

The Ingress Controller

- Specialized load balancer for k8s environments
- Manages traffic to pods in your cluster
- Like Nodes, Services etc., they are deployed using the cluster API
- There are several available besides the k8s offering

Lab 16

Rolling Deployments

- Code changes, so your pod (containers) change
- We want a stable experience for users
- k8s does this via rolling deployments
- As updates nodes become available, traffic is load balanced to ensure uptime

Blue/Green Deployments

- How they work: keep two identical environments, conventionally called blue and green, to do continuous, risk-free updates. This way, users access one while the other receives updates.
- With bare metal servers, this is expensive – one of the environments isn't used much when the other is
- k8s makes this easy – we can make a second environment, switch traffic over, then kill the first with little impact

Canary Pods for Testing and Production

- A strategy to try out a new feature only for a small segment of users or traffic (A/B testing)
- k8s makes this easy
- Use a tag to designate what pods will receive a small percentage of traffic
- Also, allows smoke testing on small scale before “opening floodgates”

NetworkPolicy and SecurityContext

- NetworkPolicy can be used to manage access within a Cluster
- Work like firewall rules and use label selectors for blocking traffic
- Provides ingress rules and egress rules to manage flow
- Rely on network implementation in Cluster for enforcement

NetworkPolicy and SecurityContext

- SecurityContext provides an option for container security
- Alternative to containers running as root or super user account
- Offers more fine-grained control than at Pod level, enabling explicit addition and removal of Linux kernel options
- Must remain aware of potential breaks in apps caused by security config

Lab 17

Lab 18

Demo – Blue/Green Deployments

Helm

Helm

- No real concept of “applications” in k8s
- Each “application” is a collection of YAML files that define the required resources
- Helm provides the concept of a repository in which the set of YAML files for an “application” can be stored and pulled/executed together
- Supports concept of public and private repositories
- Separate install https://helm.sh/docs/intro/install/

Helm

- To use a repository, add using *helm repo add <local name> <url>*
- Update local repository cache using *helm repo update*
- Use *helm search repo <app name> --versions* to search for app in the repository cache

Helm

- Client-side tool that uses same connection info as Kubernetes
- Extends YAML to provide support for things like parameters
- An “application” package is a *chart*
- Charts can be developed/deployed locally or published to a repository
- Installing a chart creates a *release*
- Each release has a name – can install multiple instances of same app by using different names

Helm

- Manifests in a chart can support parameters
- Allows varying specific settings at time of install
- Each chart contains a set of default values

Helm

- Use *helm show values* to see defined default values for parameters
- Use *helm install* to install a chart
- Use *--set* flags on *install* to change parameters from defaults
- *helm ls* will show you installed releases

Helm

- Can create your own charts
- *helm create* scaffolds a boilerplate structure
- Provides Chart.yaml, values.yaml, templates folder
- Use {{ }} to templatize your manifest with a parameter
- Can use *helm lint* to validate contents of chart

Role-Based Access Control (RBAC)

RBAC (Role-Based Access Control)

- System of “least privilege”
- Allow only, no deny – absence of a permission implies denial
- RBAC is used to grant permissions on resources in k8s
- All resources can be secured
- Permissions get applied to a subject (user, service account, group) but not directly

RBAC (Role-Based Access Control)

- Set permissions in a role
- Apply a role to one or subjects using a role binding
- Role and RoleBinding operate at the level of namespace
- ClusterRole and ClusterRoleBinding operate at the cluster level

RBAC (Role-Based Access Control)

- Kubernetes does not authenticate end users – delegates to external identity providers
- Able to connect to your corporate provider (for example, LDAP or Active Directory)
- Cloud k8s offerings integrate with their available provider
- Can configure custom OIDC (Open ID connect) provider but not a simple task
- If using a managed k8s service, authentication will be built in

RBAC (Role-Based Access Control)

- One option to create a “user” within Kubernetes is via certificates
- Kubernetes can issue client certificates for end users requested with a username
- Certificate supplied on Kubernetes API requests is used to validate permissions to accomplish a task – attached by name
- Permissions are all additive – start with none (“least privilege”)
- Able to impersonate a user using the *--as* flag on a kubectl command

RBAC (Role-Based Access Control)

- Common approach is to start with predefined cluster roles and bind to one or more subjects for a specific namespace
- Role bindings provide a layer of abstraction between users and the roles they have

Demo – Bringing it All Together

Thank you!

If you have additional questions,
please reach out to me at:
asanders@gamuttechnologysvcs.com

DevelopIntelligence
A PLURALSIGHT COMPANY