Kubernetes & Operations

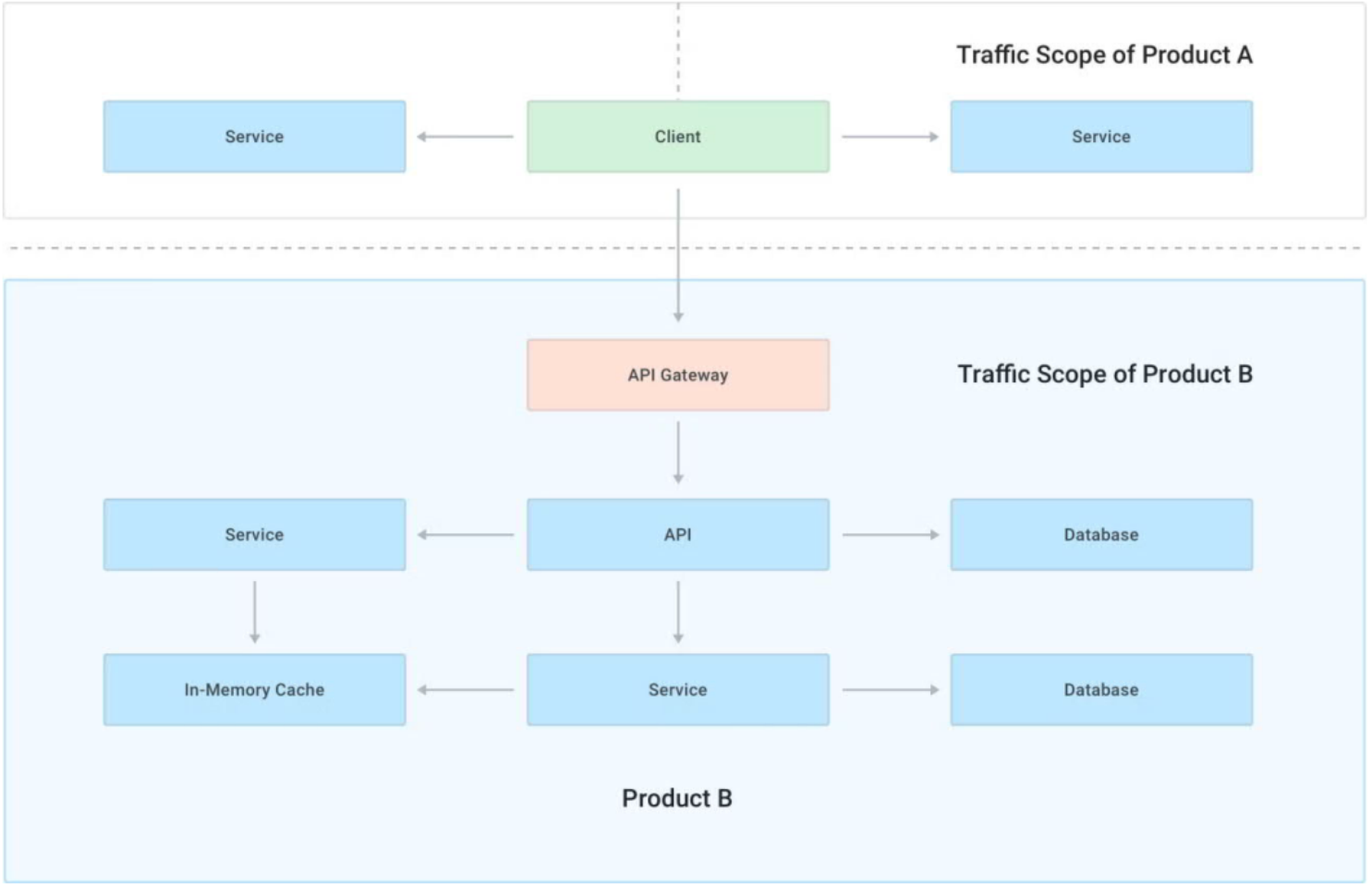
- ***Kubernetes & Containers***

- ECE 4150 – Spring 2024
- Vijay Madisetti

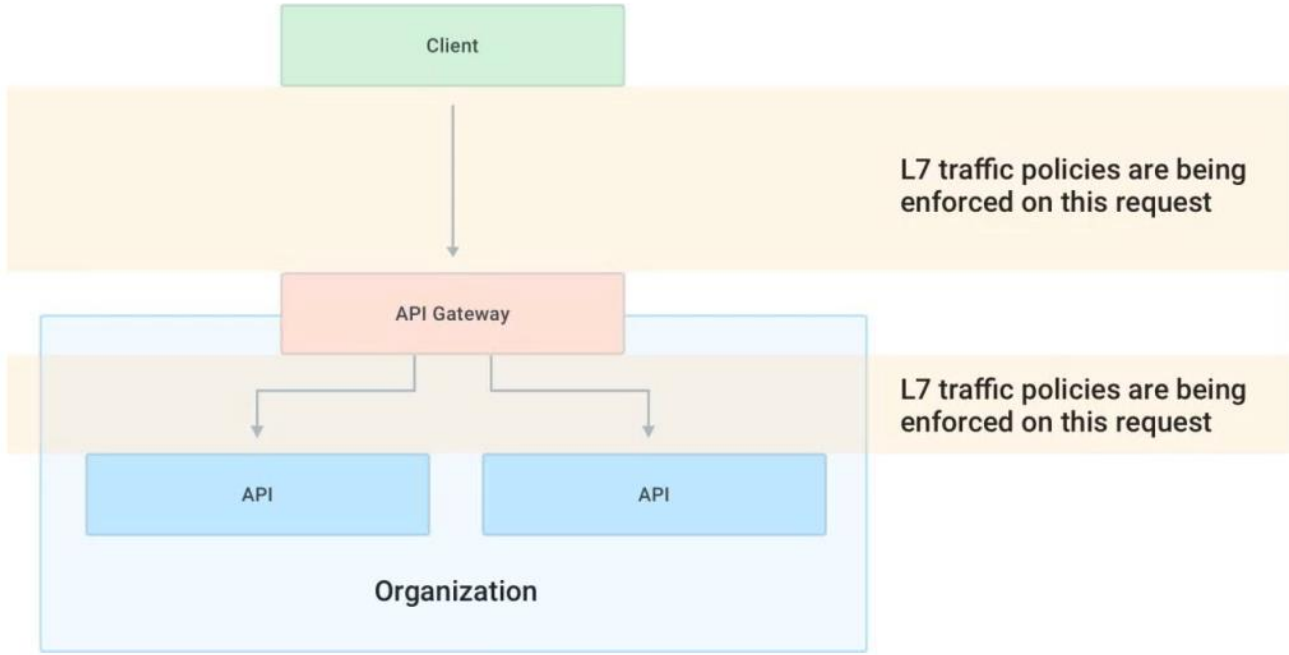
Service Mesh – Some More Details

Difference between API Gateway and Service Mesh

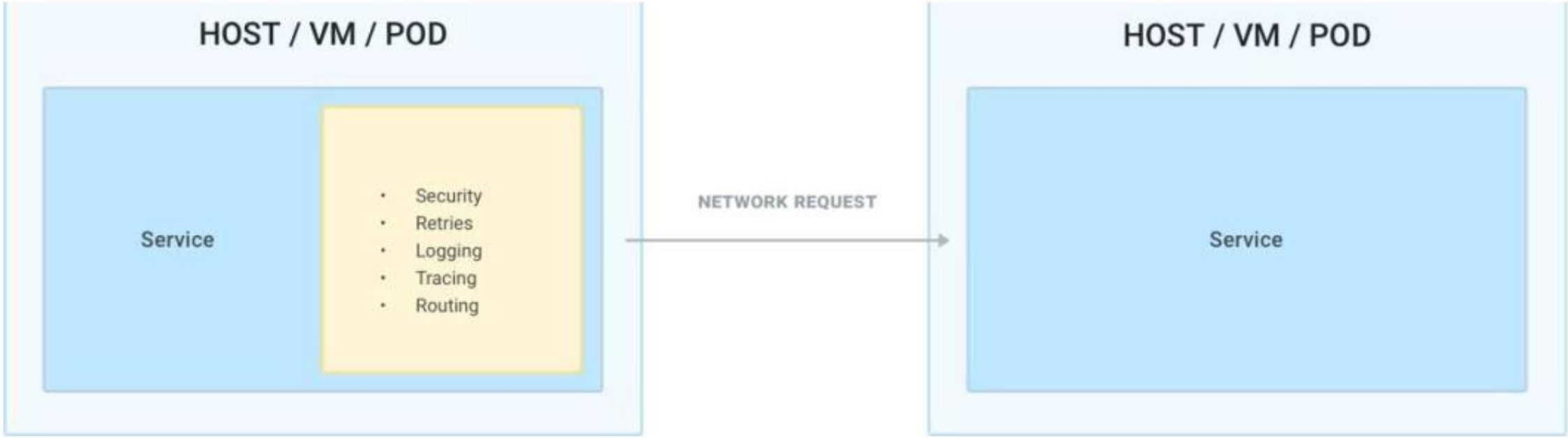
What is an API Gateway



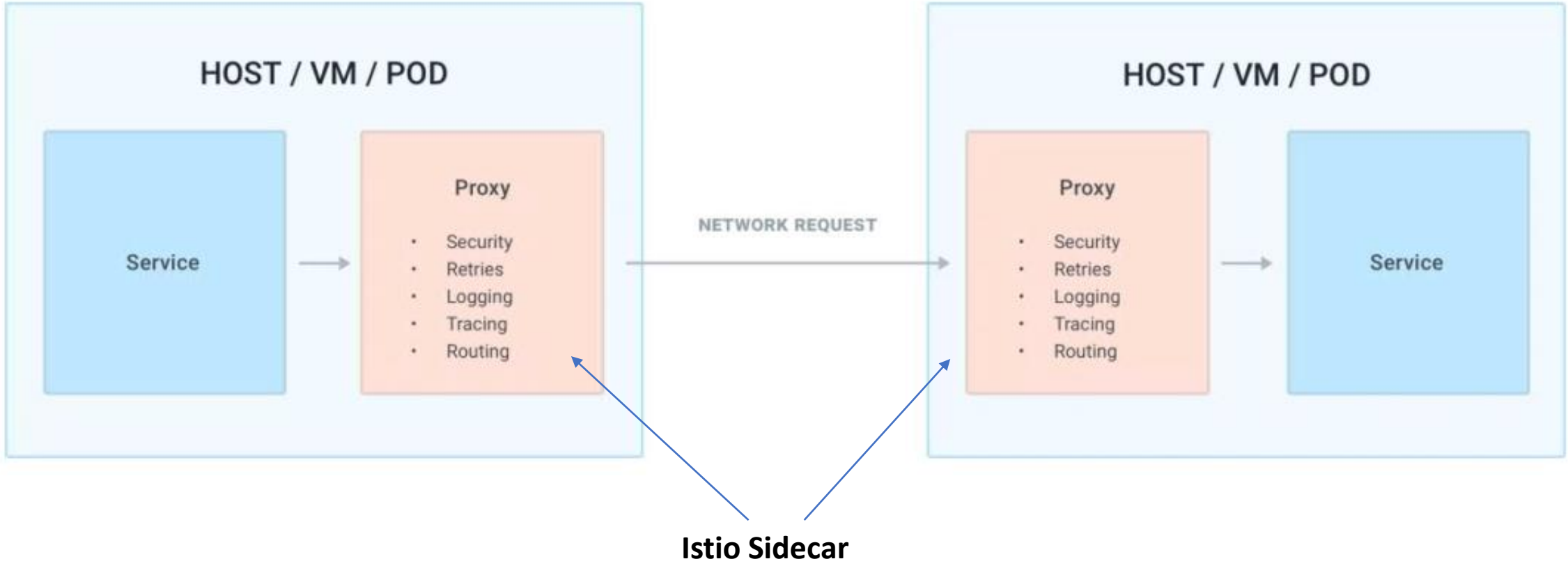
API Gateway is useful



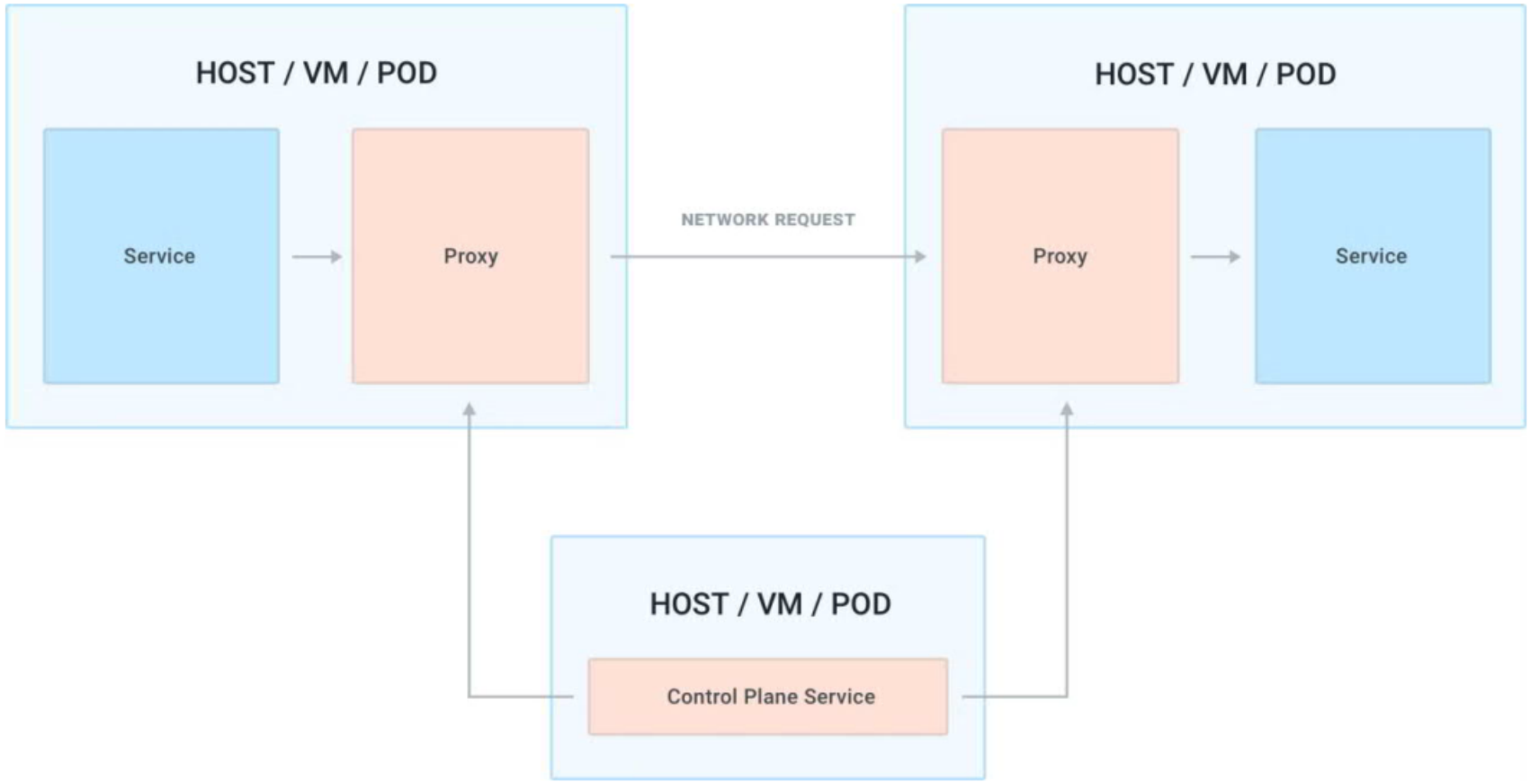
Prior to Service Meshes – Services had to talk to Services in a Custom Manner



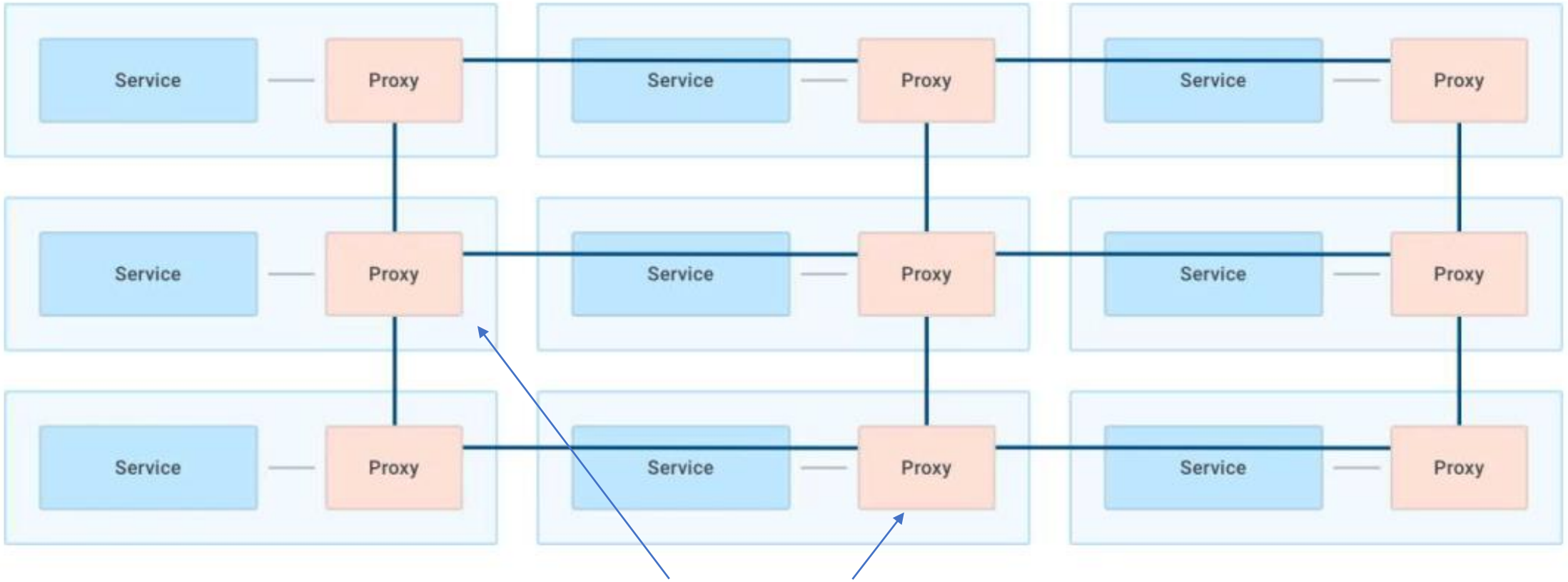
Additional Work Writing the Proxy



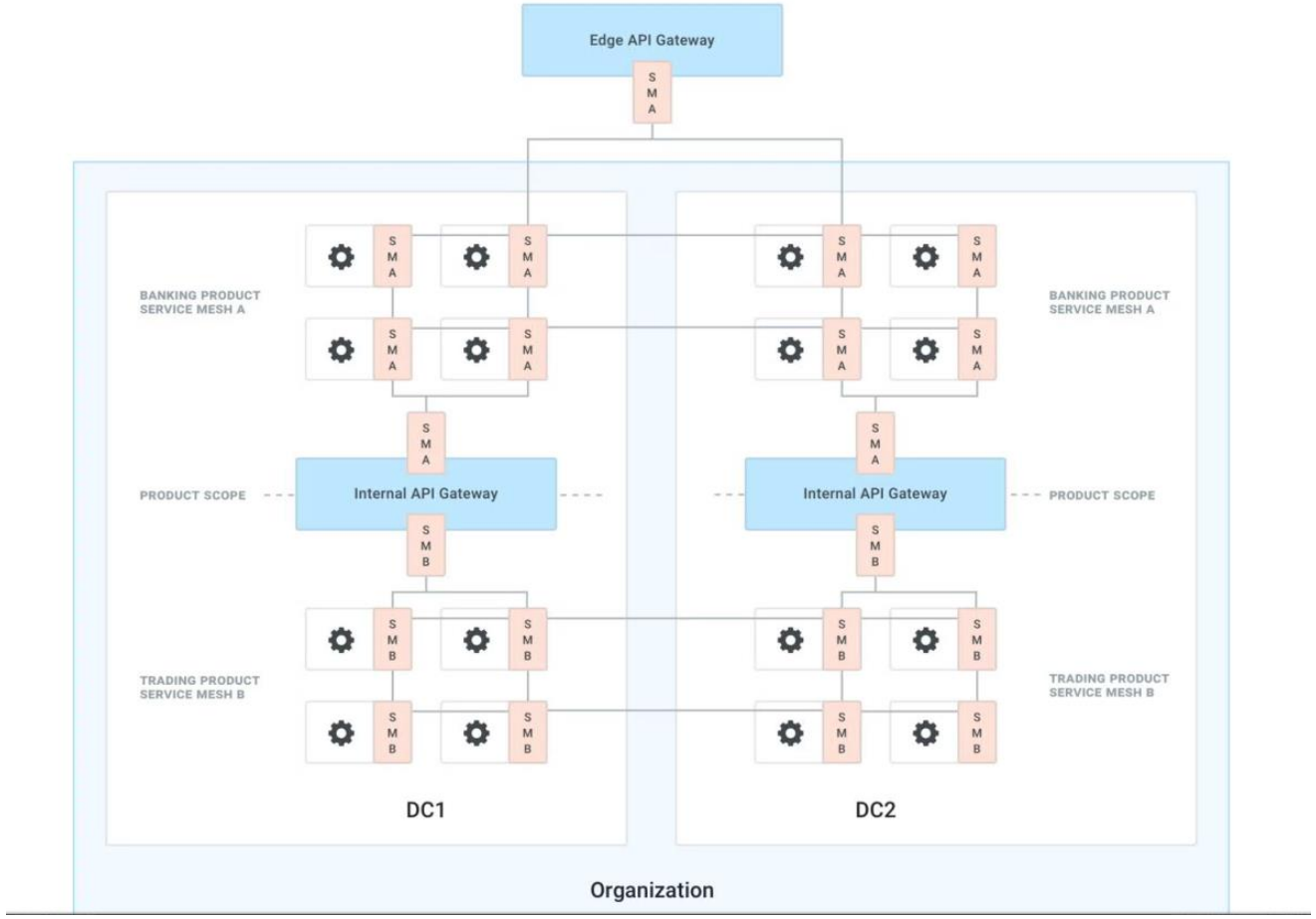
Add an Entire Service Mesh – Like a Set of Roads & Traffic Control



Service Mesh



Service Mesh as Used in a Large Bank

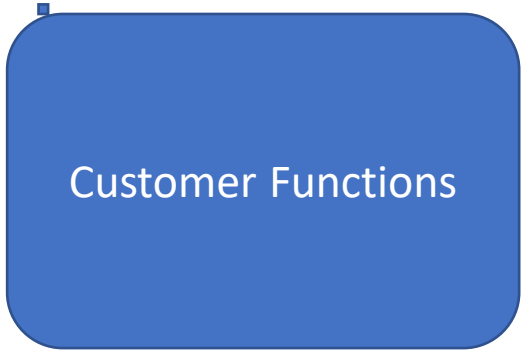


Service Mesh can use Layers ***4-7***, while API Gateway only uses Layer 7 (Application Layer)

Many Cloud Business Models

Lambda Services

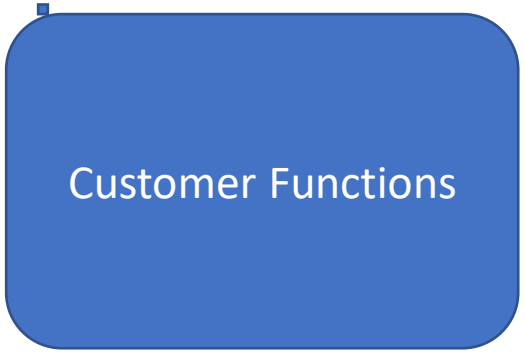
Lab 1



AWS Manages the Containers and Their Scheduling and Hides the Details from Customer
Customer is charged per Function Call

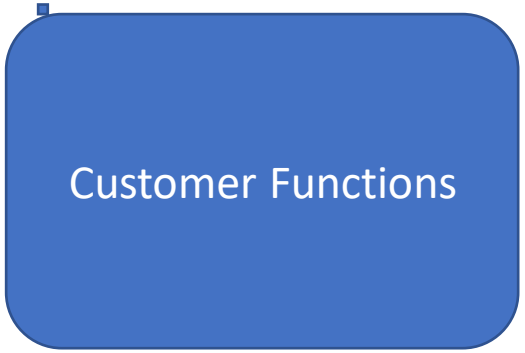
AWS Fargate

Lab 2 with help from Amazon



Amazon (AWS Fargate) Manages the Containers and Mapping to Services & Service Mesh – Customer is Charged for the Servers and their Usage

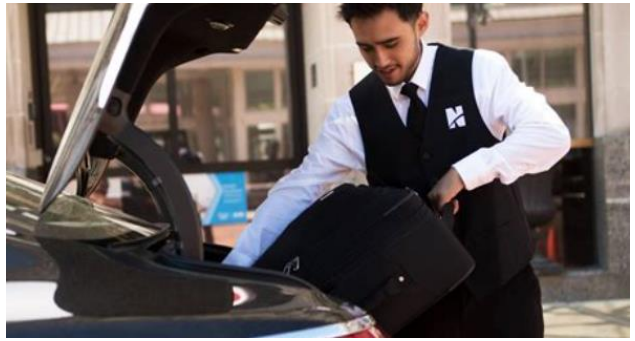
Kubernetes Model



Customer Manages Kubernetes Manages the Containers and Mapping to Services & Service Mesh – Customer is Charged for the Servers and their Usage

Cloud Models

Lambda Services Model



Valet Parking

Lab 2 – Provision your own EC2 servers

Rent U-Haul

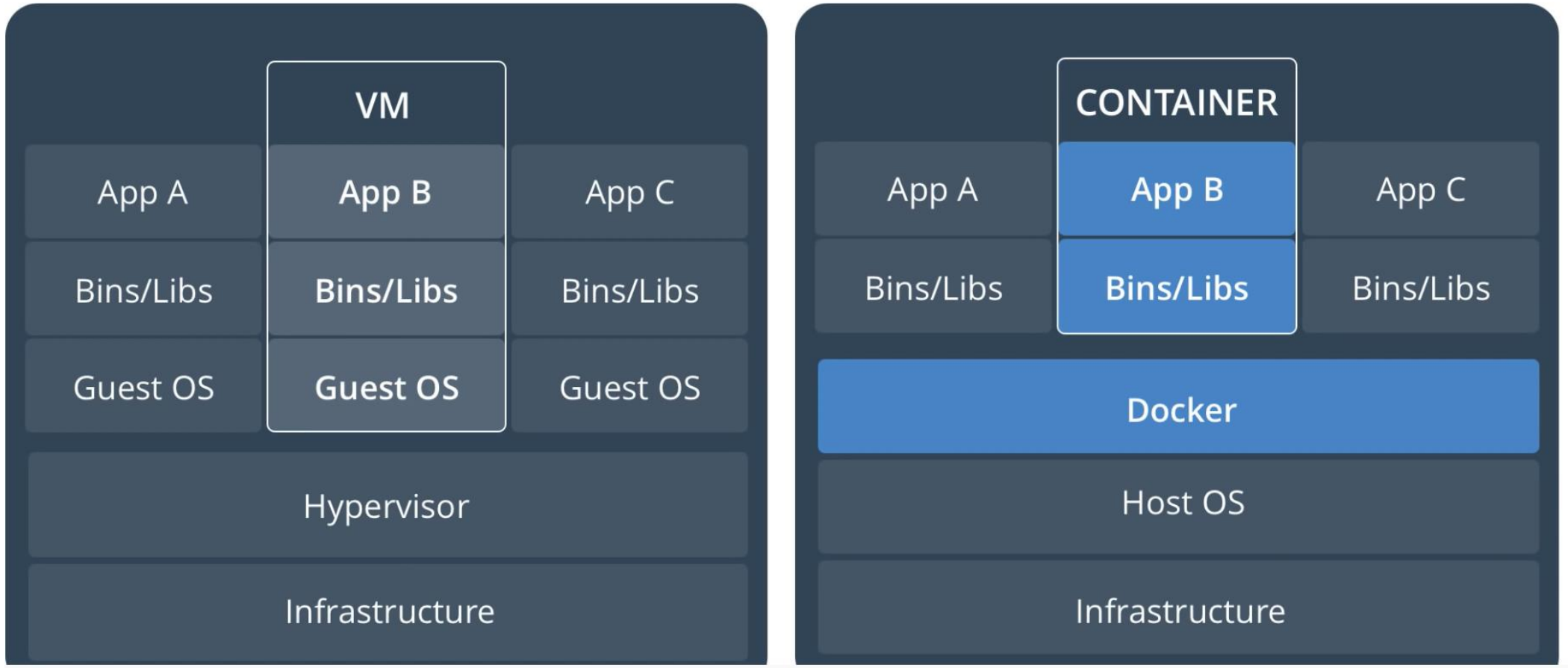
CONTAINER MODEL OF CLOUD COMPUTING

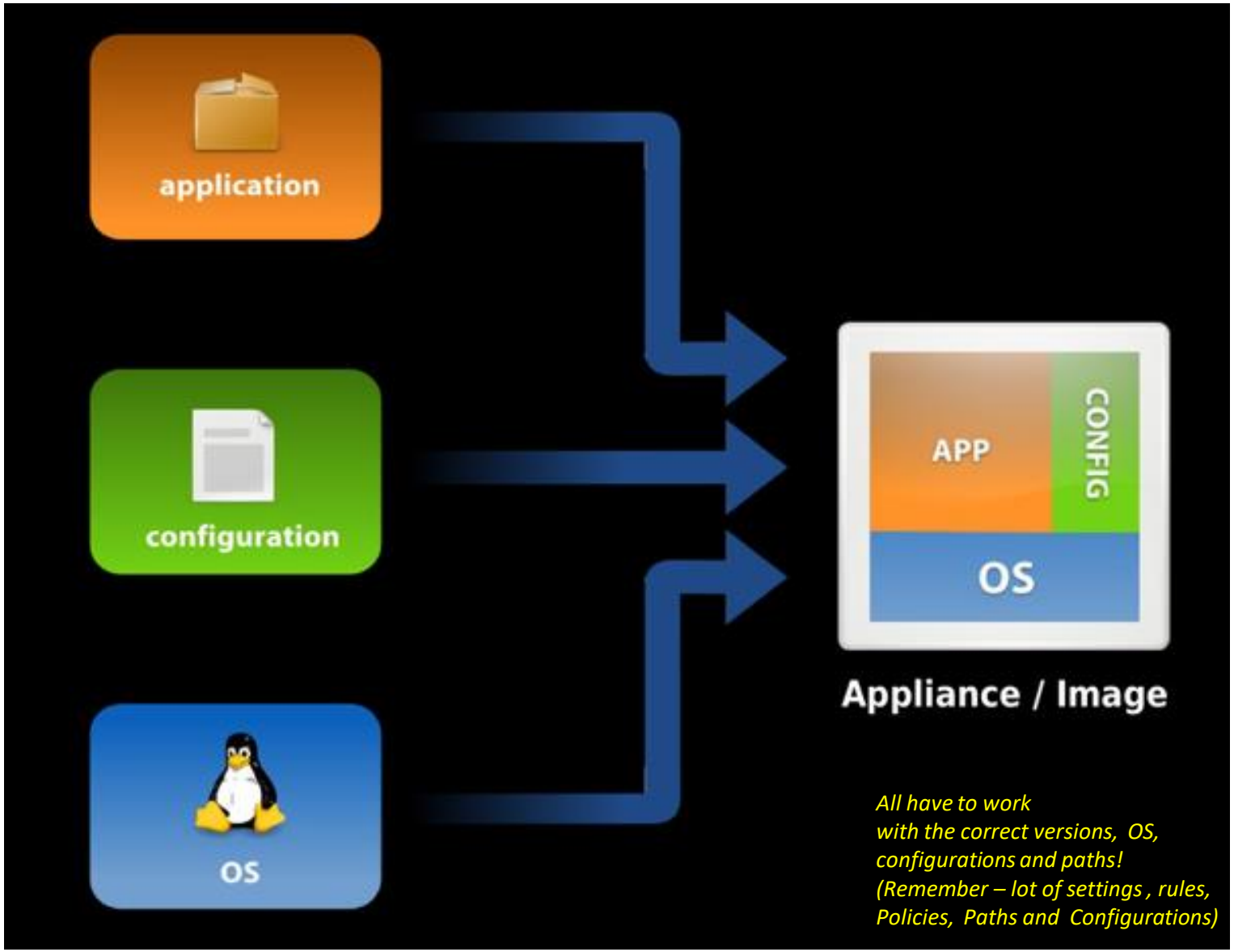
DOCKER

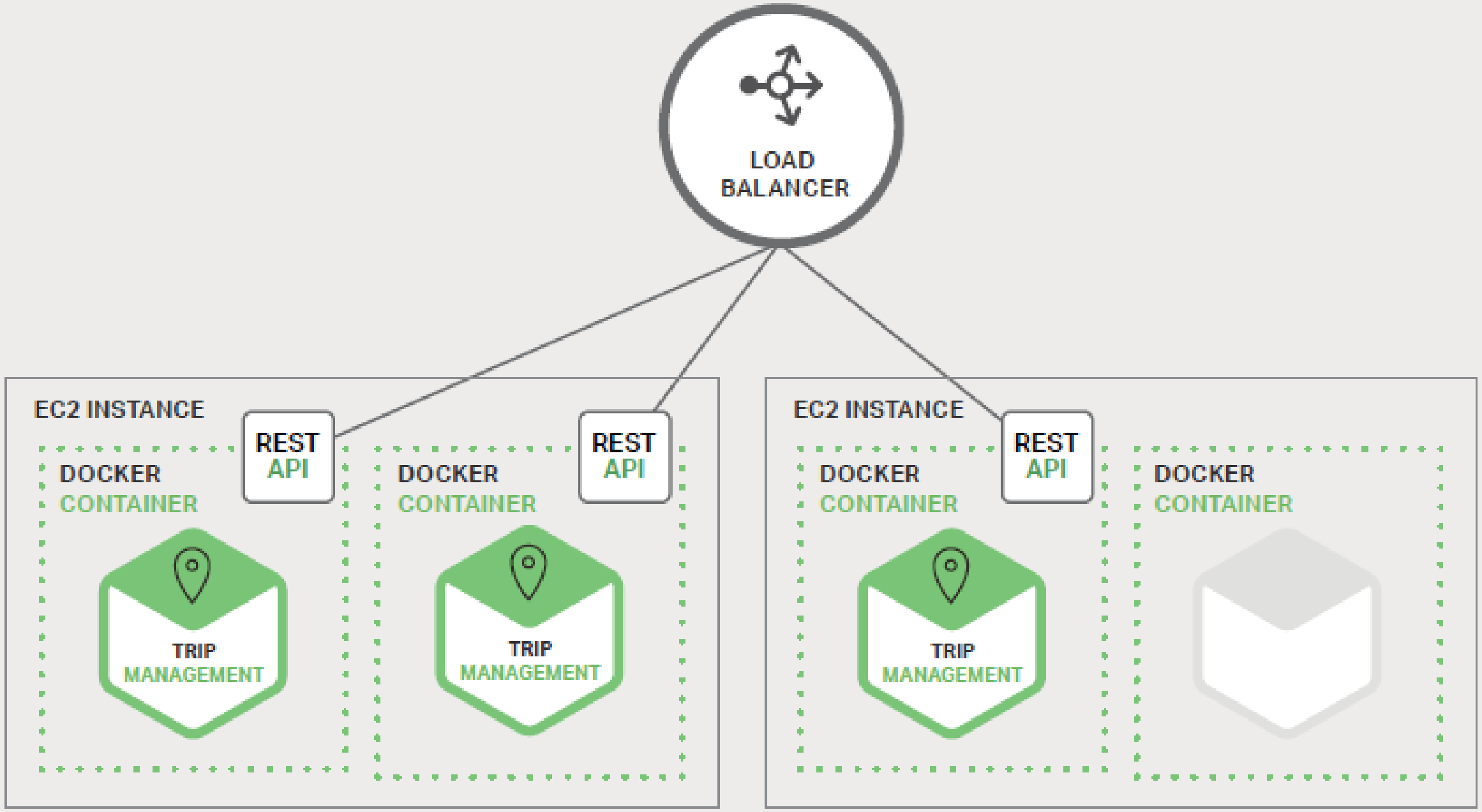
KUBERNETES

Kubernetes

Docker vs VM


VM
App A
App B
App C
Bins/Libs
Bins/Libs
Bins/Libs
Guest OS
Guest OS
Guest OS
Hypervisor
Infrastructure
CONTAINER
App A
App B
App C
Bins/Libs
Bins/Libs
Bins/Libs
Docker
Host OS
Infrastructure


application
configuration
OS
APP
CONFIG
OS
Appliance / Image
All have to work
with the correct versions, OS,
configurations and paths!
(Remember – lot of settings , rules,
Policies, Paths and Configurations)


LOAD
BALANCER
EC2 INSTANCE
REST
API
DOCKER
CONTAINER
TRIP
MANAGEMENT
REST
API
DOCKER
CONTAINER
TRIP
MANAGEMENT
EC2 INSTANCE
REST
API
DOCKER
CONTAINER
TRIP
MANAGEMENT
DOCKER
CONTAINER

- The four required files to build the docker image are as follows:

```
    Directory: C:\Users\vkm2019asus\docker3\webapp_updated


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
------         1/13/2023   7:30 PM            127 docker-compose.yml
------         1/13/2023  10:01 AM            113 Dockerfile
------         1/13/2023   9:59 AM             12 requirements.txt
------         1/14/2023   6:25 AM            589 webapp.py
```

```
113 Jan 13 20:31 Dockerfile
127 Jan 14 06:00 docker-compose.yml
 12 Jan 13 20:29 requirements.txt
654 Jan 14 06:23 webapp.py
```

The Dockerfile specifies all the commands required to assemble a docker image

```
vkm@76Creekside-2022-Laptop:/mnt/c/Users/vkm2019asus/docker1$ more Dockerfile
From python:3.4-alpine
ADD . /code
WORKDIR /code
RUN pip install -r requirements.txt
CMD ["python", "webapp.py"]
```

A Dockerfile is a simple text file that contains the commands a user could call to assemble an image.

E.g., Flask app Python code along with requirements file

Image built from Dockerfile

web_app docker container

Frequently used Dockerfile commands

Command	Overview
FROM	Specify base image
RUN	Execute specified command
ENTRYPOINT	Specify the command to execute the container
CMD	Specify the command at the time of container execution (can be overwritten)
COPY	Simple copy of files / directories from host machine to container image
ADD	COPY + unzip / download from URL (**not recommended**)
ENV	Add environment variables
EXPOSE	Open designated port
WORKDIR	Change current directory
MAINTAINER	**deprecated**
	now `LABEL maintainer="maintainer@example.com"` should be specified as

```
113 Jan 13 20:31 Dockerfile
127 Jan 14 06:00 docker-compose.yml
 12 Jan 13 20:29 requirements.txt
654 Jan 14 06:23 webapp.py
```

The **requirements.txt** specifies all the packages required to assemble a docker image

```
vkm@76Creekside-2022-Laptop:/mnt/c/Users/vkm2019asus/docker1$ more requirements.txt
flask
redis
```

```
113 Jan 13 20:31 Dockerfile
127 Jan 14 06:00 docker-compose.yml
 12 Jan 13 20:29 requirements.txt
654 Jan 14 06:23 webapp.py
```

The **docker-compose.yml** defines the services used with a multi-container docker applications

```
vkm@76Creekside-2022-Laptop:/mnt/c/Users/vkm2019asus/docker1$ more docker-compose.yml
version: '3.9'
services:
   web:
     build: .
     ports:
        - "8000:5000"
   redis:
     image: "redis:alpine"
```

Docker Compose

- is a tool for defining and running multi-container Docker applications.
- define the services that make up your app in docker-compose.yml so they can be run together in an isolated environment.
- get an app running in one command by just running docker-compose up

Docker-compose quick reference

Define services, networks and volumes

File

structure

```
services:
   container1:
      properties: values

   container2:
      properties: values

networks:
   network:

volumes:
   volume:
```

Types

value

```
key: value
```

array

```
key:
   - value
   - value
```

dictionary

```
master:
   key: value
   key: value
```

Properties

build

build image from dockerfile in specified directory

```
container:
   build: ./path
   image: image-name
```

image

use specified image

```
image: image-name
```

container_name

define container name to access it later

```
container_name: name
```

volumes

define container volumes to persist data

```
volumes:
   - /path:/path
```

command

override start command for the container

```
command: execute
```

environment

define env variables for the container

```
environment:
   KEY: VALUE
---
environment:
   - KEY=VALUE
```

env_file

define a env file for the container to set and override env variables

```
env_file: .env
---
env_file:
 - .env
```

restart

define restart rule (no, always, on-failure, unless-stopped)

```
expose:
    - "9999"
```

networks

define all networks for the container

```
networks:
   - network-name
```

ports

define ports to expose to other containers and host

```
ports:
    - "9999:9999"
```

expose

define ports to expose only to other containers

```
expose:
    - "9999"
```

network_mode

define network driver (bridge, host, none, etc.)

```
network_mode: host
```

depends_on

define build, start and stop order of container

```
depends_on:
    - container-name
```

Other

idle container

send container to idle state > container will not stop

```
command: tail -f /dev/null
```

named volumes

create volumes that can be used in the volumes property

```
services:
  container:
    image: image-name
    volumes:
      - data-
volume:/path/to/dir

volumes:
  data-volume:
```

networks

create networks that can be used in the networks property

```
networks:
   frontend:
      driver: bridge
```

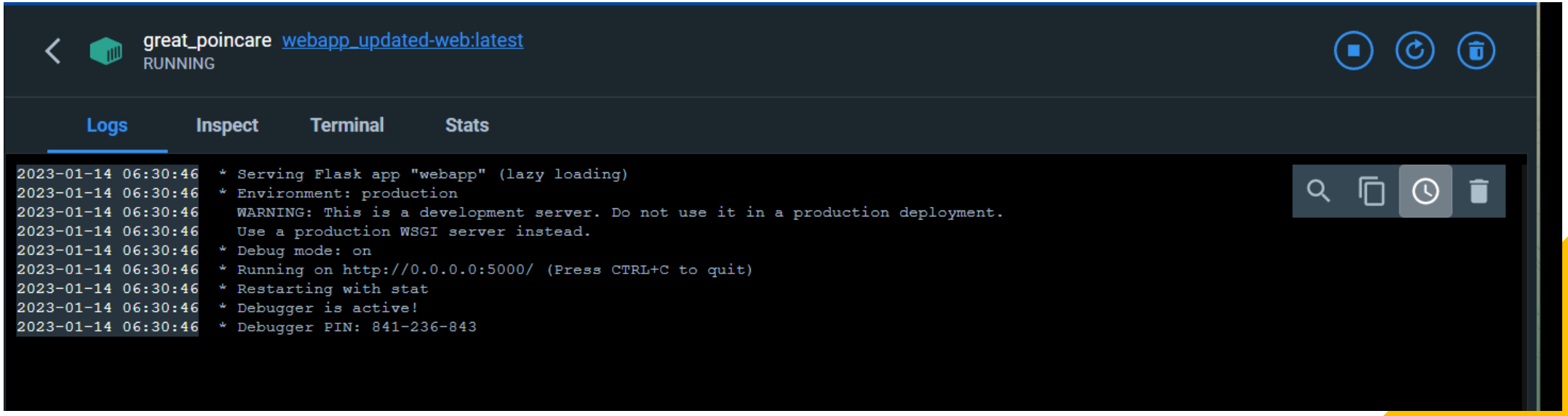
https://**www.docker.com**/blog/how-to-dockerize-your-python-applications/

Webapp.py

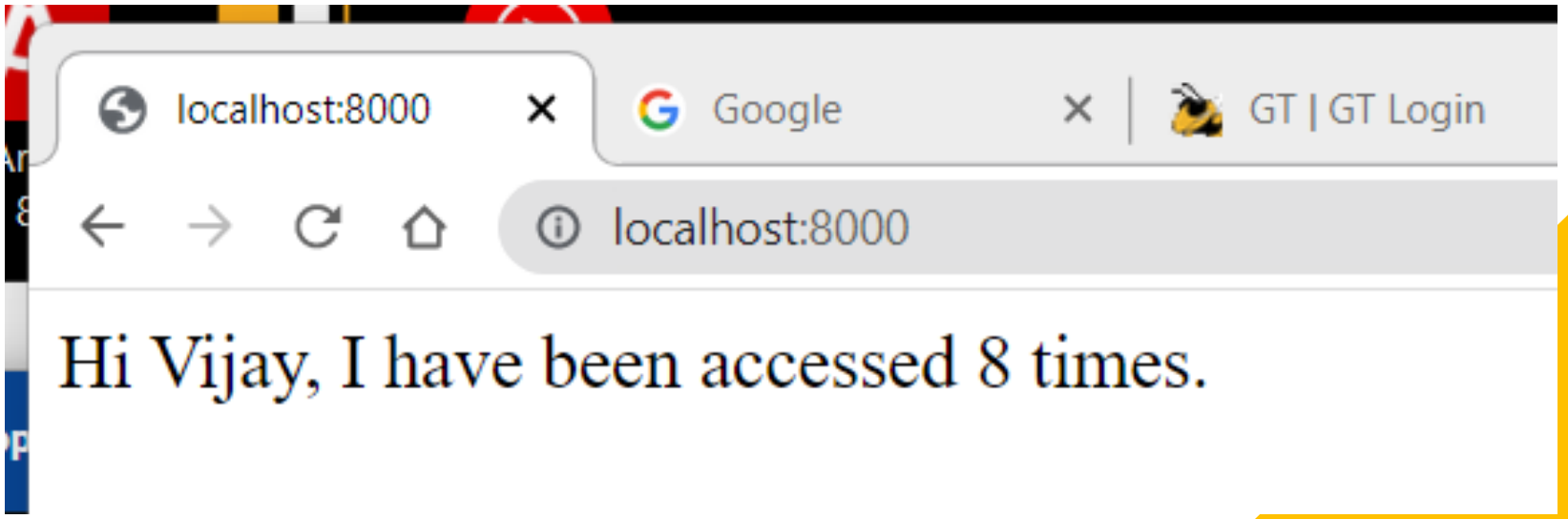
webapp.py - Notepad

File Edit View

```
import time
import redis
from flask import Flask

app = Flask(__name__)
cache = redis.Redis(host='redis', port=6379)

def get_hit_count():
    retries = 5
    while True:
        try:
            return cache.incr('hits')
        except redis.exceptions.ConnectionError as exc:
            if retries ==0:
                raise exc
            retries -= 1
            time.sleep(0.5)

@app.route('/')
def hello():
    count = get_hit_count()
    return 'Hi Vijay, I have been accessed {} times.'.format(count)

if __name__ =="__main__":
    app.run(host="0.0.0.0", debug=True)
```


great_poincare webapp_updated-web:latest
RUNNING
Logs
Inspect
Terminal
Stats
2023-01-14 06:30:46 * Serving Flask app "webapp" (lazy loading)
2023-01-14 06:30:46 * Environment: production
2023-01-14 06:30:46 WARNING: This is a development server. Do not use it in a production deployment.
2023-01-14 06:30:46 Use a production WSGI server instead.
2023-01-14 06:30:46 * Debug mode: on
2023-01-14 06:30:46 * Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
2023-01-14 06:30:46 * Restarting with stat
2023-01-14 06:30:46 * Debugger is active!
2023-01-14 06:30:46 * Debugger PIN: 841-236-843


localhost:8000
Hi Vijay, I have been accessed 1 times.


localhost:8000
Google
GT | GT Login
localhost:8000
Hi Vijay, I have been accessed 8 times.

Cheatsheet for Docker CLI

Run a new Container

Start a new Container from an Image
```
docker run IMAGE
docker run nginx
```

...and assign it a name
```
docker run --name CONTAINER IMAGE
docker run --name web nginx
```

...and map a port
```
docker run -p HOSTPORT:CONTAINERPORT IMAGE
docker run -p 8080:80 nginx
```

...and map all ports
```
docker run -P IMAGE
docker run -P nginx
```

...and start container in background
```
docker run -d IMAGE
docker run -d nginx
```

...and assign it a hostname
```
docker run --hostname HOSTNAME IMAGE
docker run --hostname srv nginx
```

...and add a dns entry
```
docker run --add-host HOSTNAME:IP IMAGE
```

...and map a local directory into the container
```
docker run -v HOSTDIR:TARGETDIR IMAGE
docker run -v ~/:/usr/share/nginx/html nginx
```

...but change the entrypoint
```
docker run -it --entrypoint EXECUTABLE IMAGE
docker run -it --entrypoint bash nginx
```

Manage Containers

Show a list of running containers
```
docker ps
```

Show a list of all containers
```
docker ps -a
```

Delete a container
```
docker rm CONTAINER
docker rm web
```

Delete a running container
```
docker rm -f CONTAINER
docker rm -f web
```

Delete stopped containers
```
docker container prune
```

Stop a running container
```
docker stop CONTAINER
docker stop web
```

Start a stopped container
```
docker start CONTAINER
docker start web
```

Copy a file from a container to the host
```
docker cp CONTAINER:SOURCE TARGET
docker cp web:/index.html index.html
```

Copy a file from the host to a container
```
docker cp TARGET CONTAINER:SOURCE
docker cp index.html web:/index.html
```

Start a shell inside a running container
```
docker exec -it CONTAINER EXECUTABLE
docker exec -it web bash
```

Rename a container
```
docker rename OLD_NAME NEW_NAME
docker rename 096 web
```

Create an image out of container
```
docker commit CONTAINER
docker commit web
```

Manage Images

Download an image
```
docker pull IMAGE[:TAG]
docker pull nginx
```

Upload an image to a repository
```
docker push IMAGE
docker push myimage:1.0
```

Delete an image
```
docker rmi IMAGE
```

Show a list of all Images
```
docker images
```

Delete dangling images
```
docker image prune
```

Delete all unused images
```
docker image prune -a
```

Build an image from a Dockerfile
```
docker build DIRECTORY
docker build .
```

Tag an image
```
docker tag IMAGE NEWIMAGE
docker tag ubuntu ubuntu:18.04
```

Build and tag an image from a Dockerfile
```
docker build -t IMAGE DIRECTORY
docker build -t myimage .
```

Save an image to .tar file
```
docker save IMAGE > FILE
docker save nginx > nginx.tar
```

Load an image from a .tar file
```
docker load -i TARFILE
docker load -i nginx.tar
```

Info & Stats

Show the logs of a container
```
docker logs CONTAINER
docker logs web
```

Show stats of running containers
```
docker stats
```

Show processes of container
```
docker top CONTAINER
docker top web
```

Show installed docker version
```
docker version
```

Get detailed info about an object
```
docker inspect NAME
docker inspect nginx
```

Show all modified files in container
```
docker diff CONTAINER
docker diff web
```

Show mapped ports of a container
```
docker port CONTAINER
docker port web
```

Show list of running containers using:
docker ps -a

```
arshdeep@pop-os:~/Downloads/webapp_updated$ docker ps -a
CONTAINER ID   IMAGE                COMMAND                  CREATED         STATUS         PORTS                                       NAMES
e3fbaa737d65   redis:alpine         "docker-entrypoint.s…"   6 seconds ago   Up 5 seconds   6379/tcp                                    webapp_updated_redis_1
e2c7f1e611cc   webapp_updated_web   "python webapp.py"       6 seconds ago   Up 5 seconds   0.0.0.0:8000->5000/tcp, :::8000->5000/tcp   webapp_updated_web_1
```

Docker images used for containers

Port 6379 for Redis in the container is mapped to the same port on the host

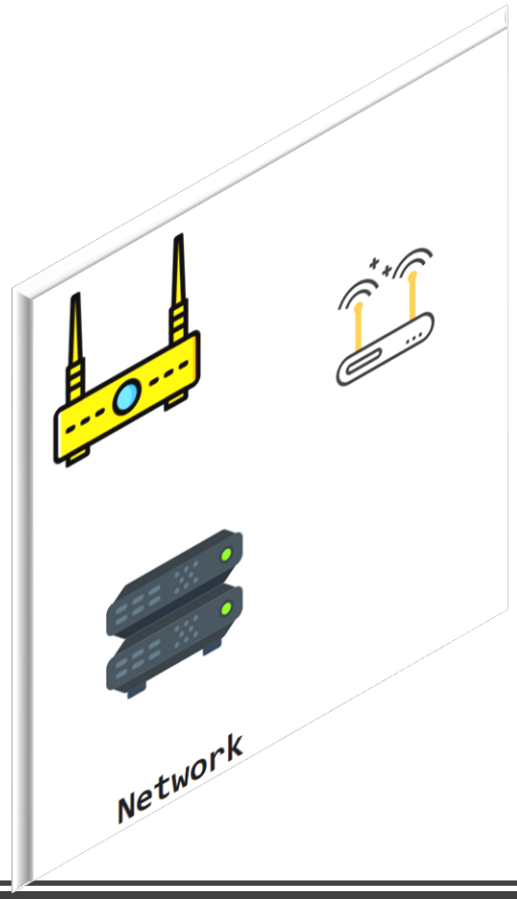
Port 5000 for Flask app in the container is mapped to port 8000 on the host

Show list of docker images

```
arshdeep@pop-os:~/Downloads/webapp_updated$ docker images
REPOSITORY            TAG           IMAGE ID       CREATED          SIZE
webapp_updated_web    latest        19635a8c4bfd   44 hours ago     84.6MB
webapp_web            latest        5b7711e3e734   44 hours ago     84.6MB
redis                 alpine        26b875a60c63   6 days ago       29.9MB
```

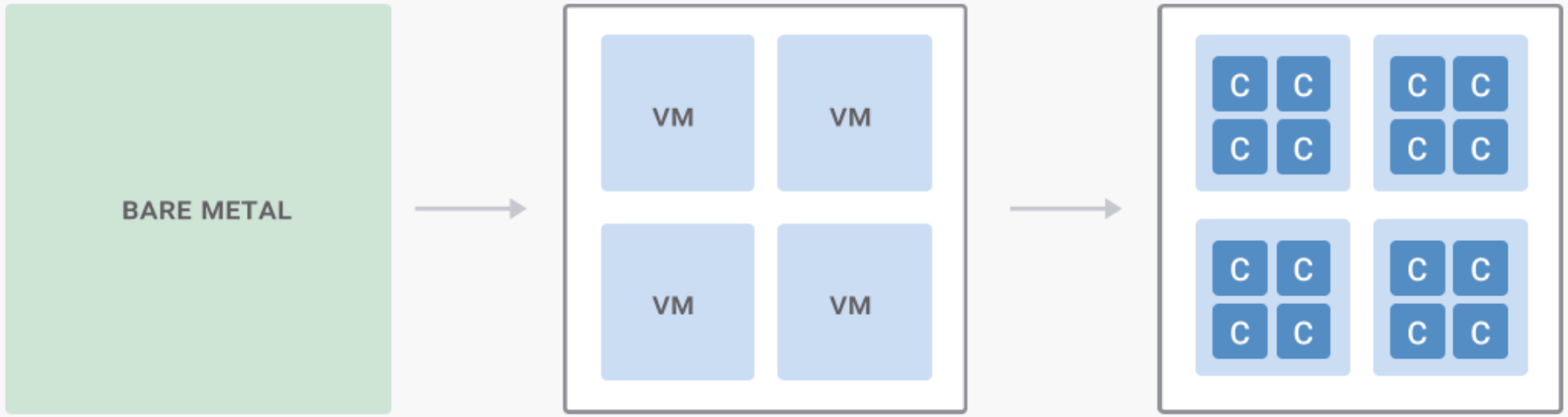
Show logs of a container

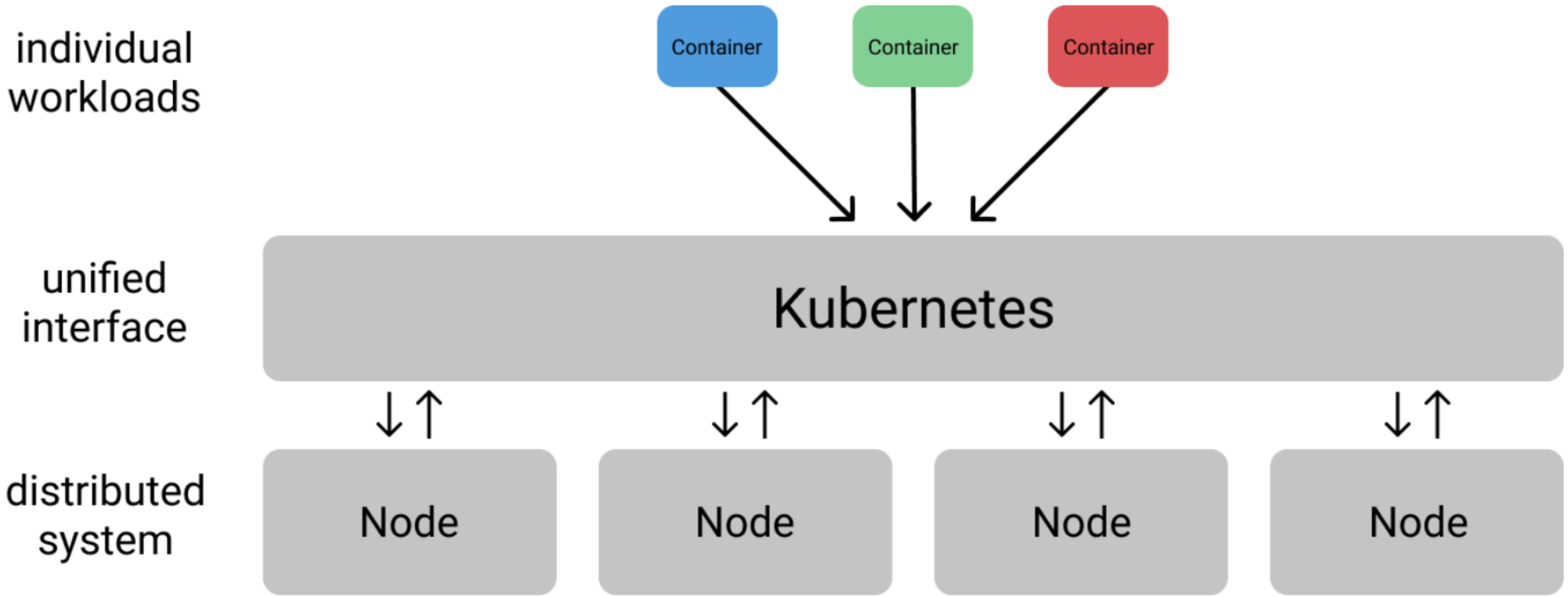
```
arshdeep@pop-os:~/Downloads/webapp_updated$ docker logs webapp_updated_web_1
 * Serving Flask app "webapp" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: on
 * Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 334-243-052
```



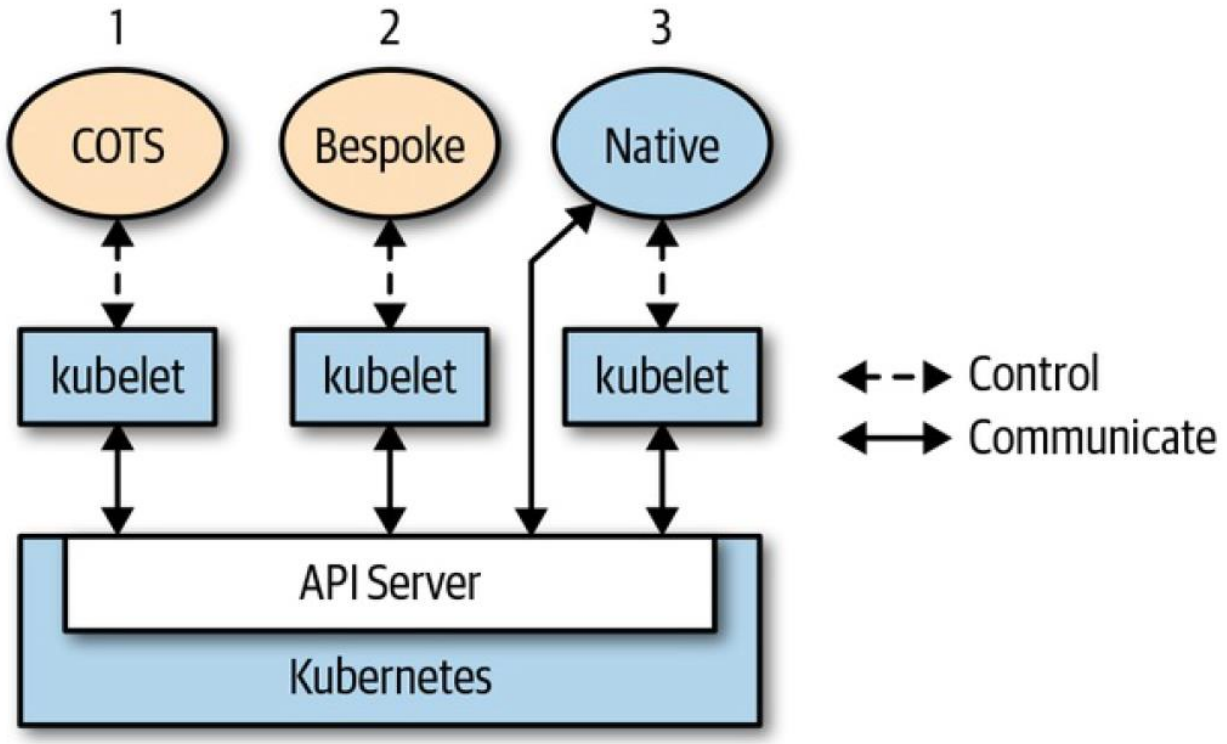
Cloud Resources are not just EC2 servers – networking and storage components are resources too that need to be managed.

Abstraction layers: From servers, to VM, to Containers



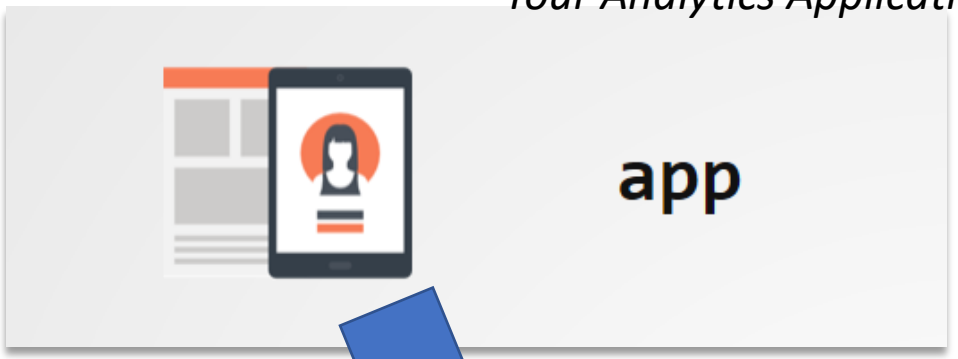

individual workloads
Container
Container
Container
unified interface
Kubernetes
distributed system
Node
Node
Node
Node

Kubernetes and Cloud Native Applications



Ecosystem for Cloud Applications

Your Analytics Application

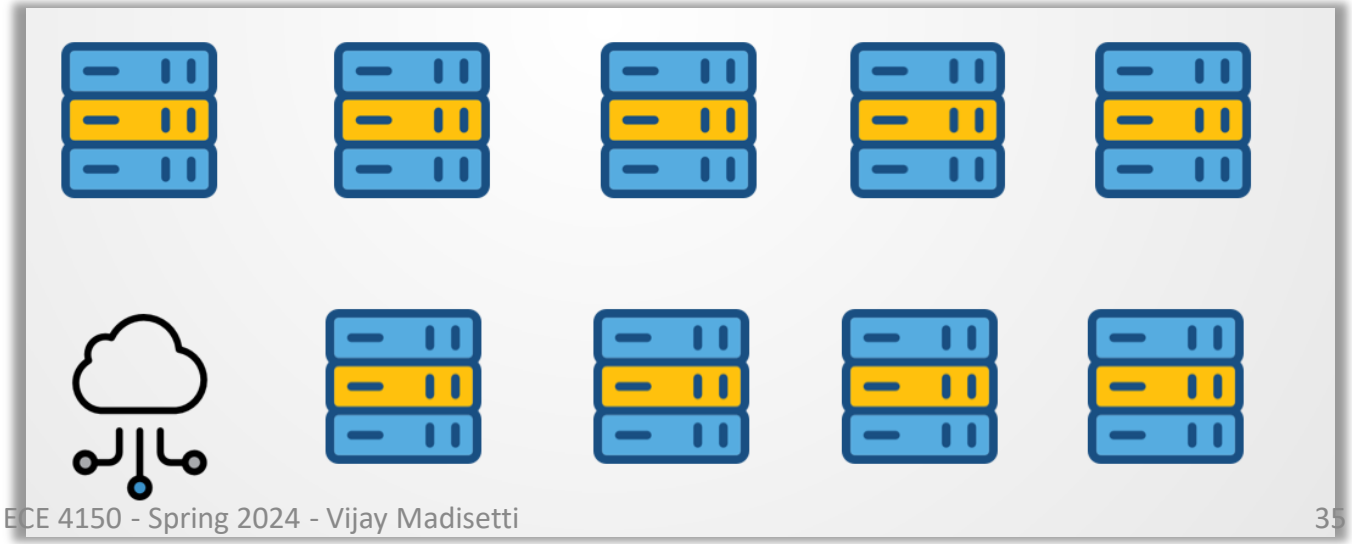


AWS

Design principles.

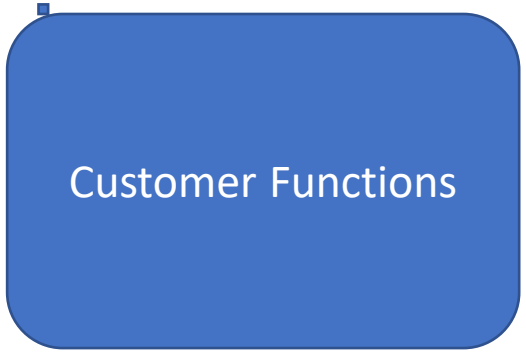
- Declarative
- Distributed
- Decoupled
- Immutable

Fault-Tolerance, Availability, Scalability, Security..



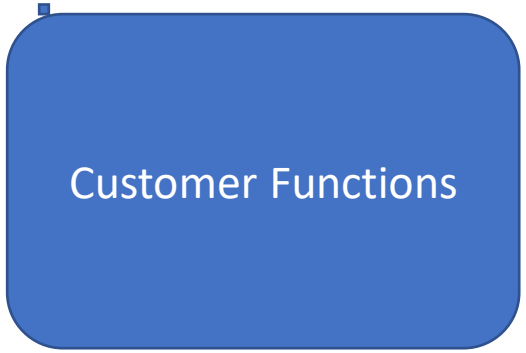
Three Business Models

Lambda Services



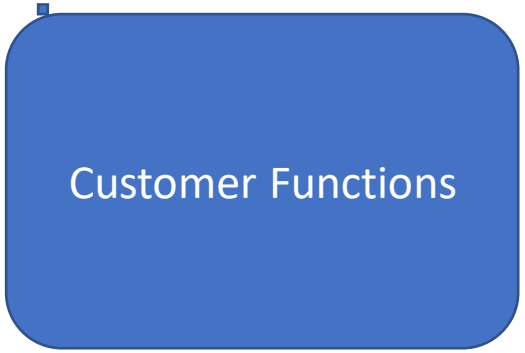
AWS Manages the Containers and Their Scheduling and Hides the Details from Customer
Customer is charged per Function Call

AWS Fargate



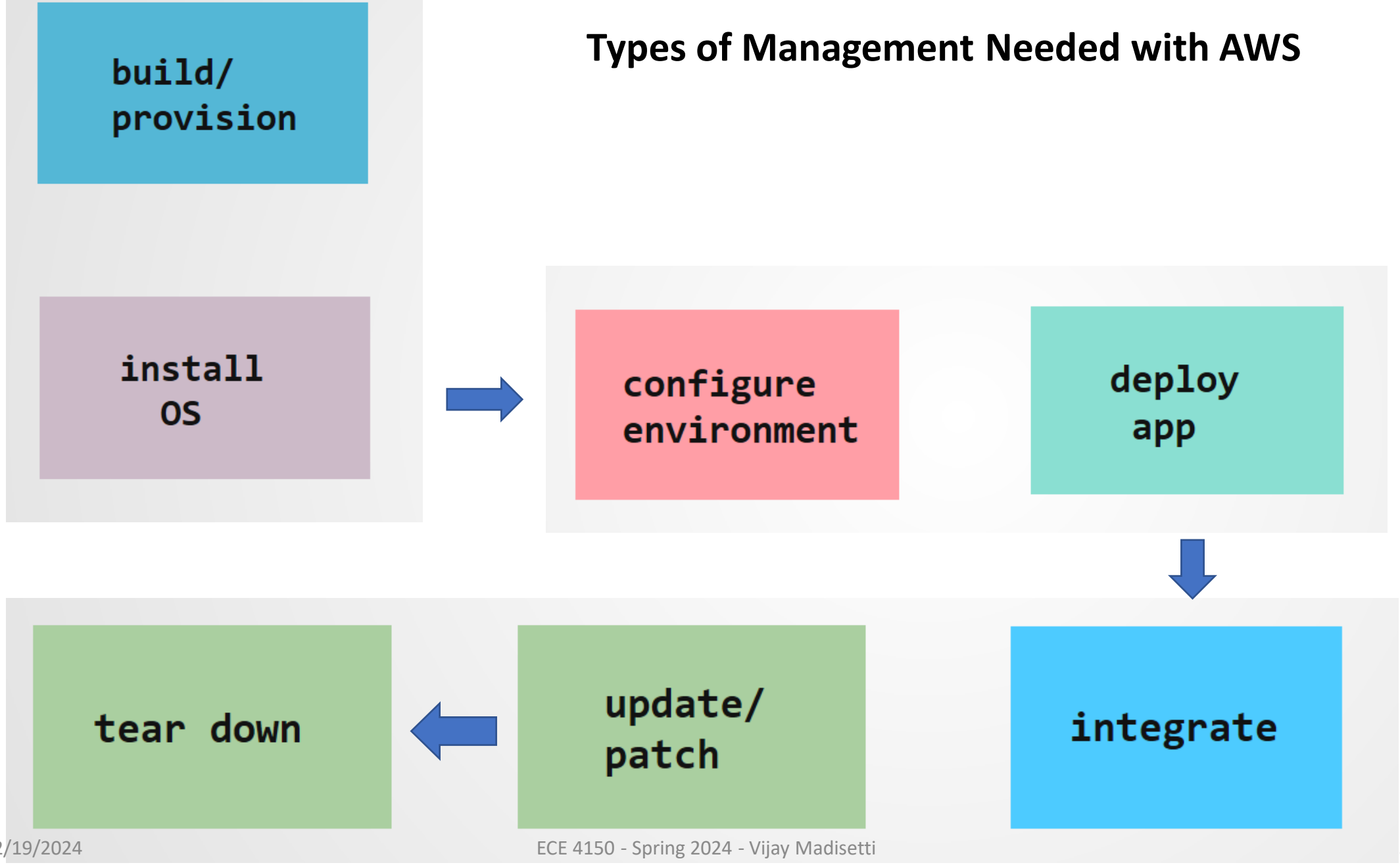
AWS Fargate Manages the Containers and Mapping to Services – Customer is Charged for the Servers and their Usage

Kubernetes Model

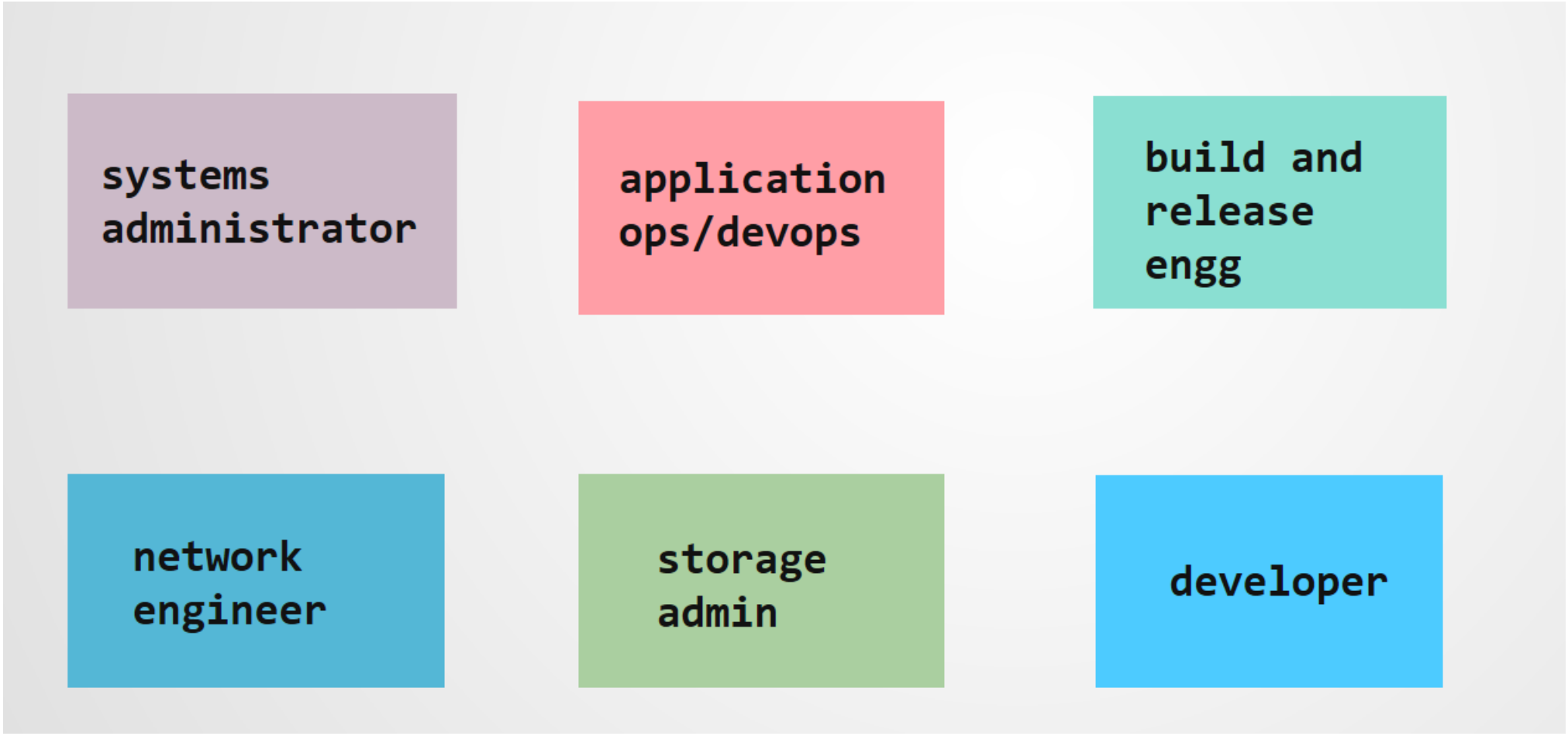


Customer Manages Kubernetes Manages the Containers and Mapping to Services – Customer is Charged for the Servers and their Usage

Types of Management Needed with AWS



Who needs to manage resources ?



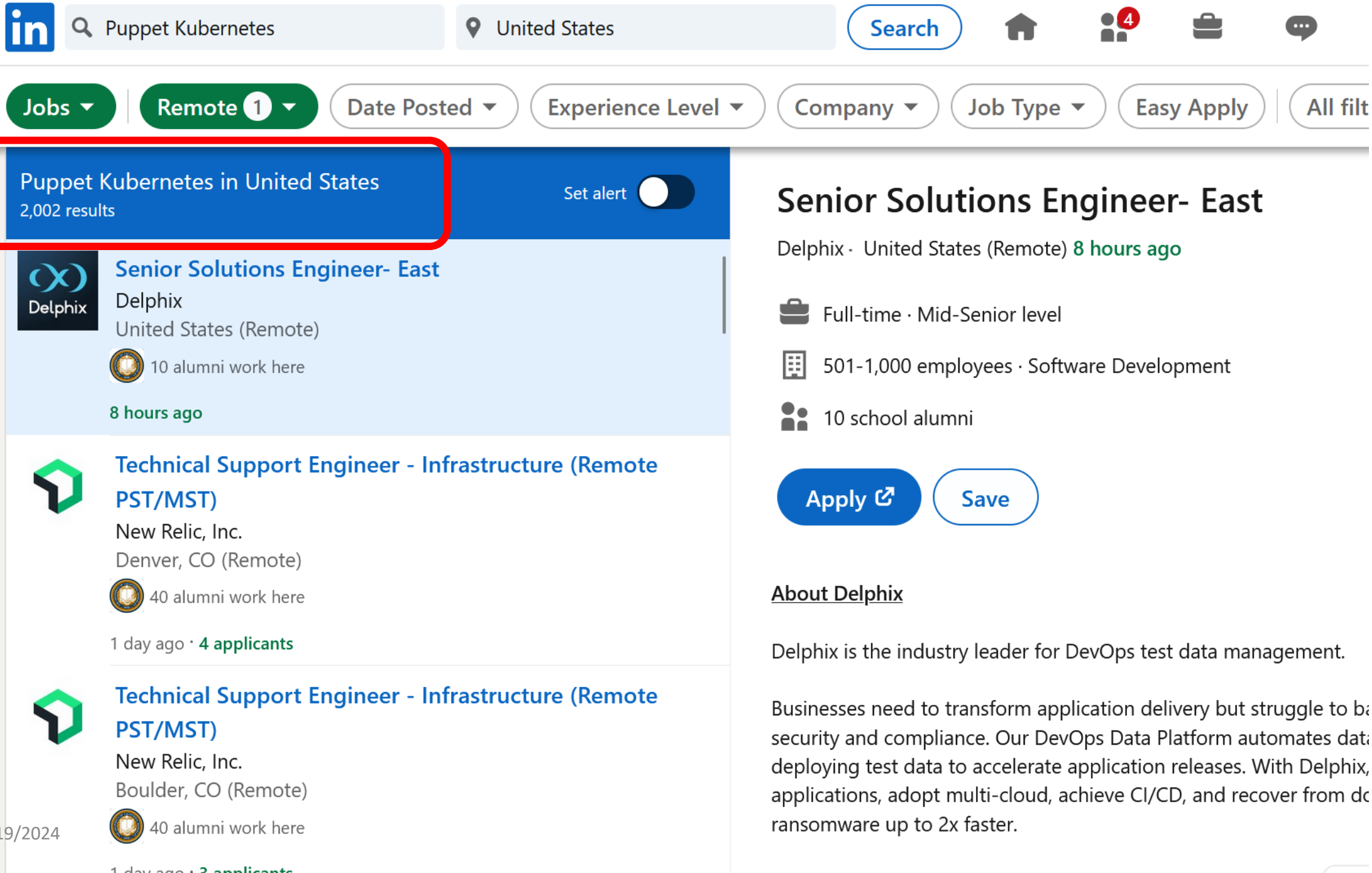
Many Ways to Manage IT Infrastucture

Manual

Scripts

Golden Images

IaaC


Puppet Kubernetes
United States
Search
Jobs
Remote 1
Date Posted
Experience Level
Company
Job Type
Easy Apply
All filt
Puppet Kubernetes in United States
2,002 results
Set alert
Senior Solutions Engineer- East
Delphix
United States (Remote)
10 alumni work here
8 hours ago
Technical Support Engineer - Infrastructure (Remote PST/MST)
New Relic, Inc.
Denver, CO (Remote)
40 alumni work here
1 day ago · 4 applicants
Technical Support Engineer - Infrastructure (Remote PST/MST)
New Relic, Inc.
Boulder, CO (Remote)
40 alumni work here
Senior Solutions Engineer- East
Delphix · United States (Remote) 8 hours ago
Full-time · Mid-Senior level
501-1,000 employees · Software Development
10 school alumni
Apply
Save
About Delphix
Delphix is the industry leader for DevOps test data management.
Businesses need to transform application delivery but struggle to ba
security and compliance. Our DevOps Data Platform automates data
deploying test data to accelerate application releases. With Delphix,
applications, adopt multi-cloud, achieve CI/CD, and recover from dc
ransomware up to 2x faster.

2/19/2024

Scripts

- Automate repeatable tasks
- Procedural Programs
- Focus on HOW

Difference between Scripts and IaaC

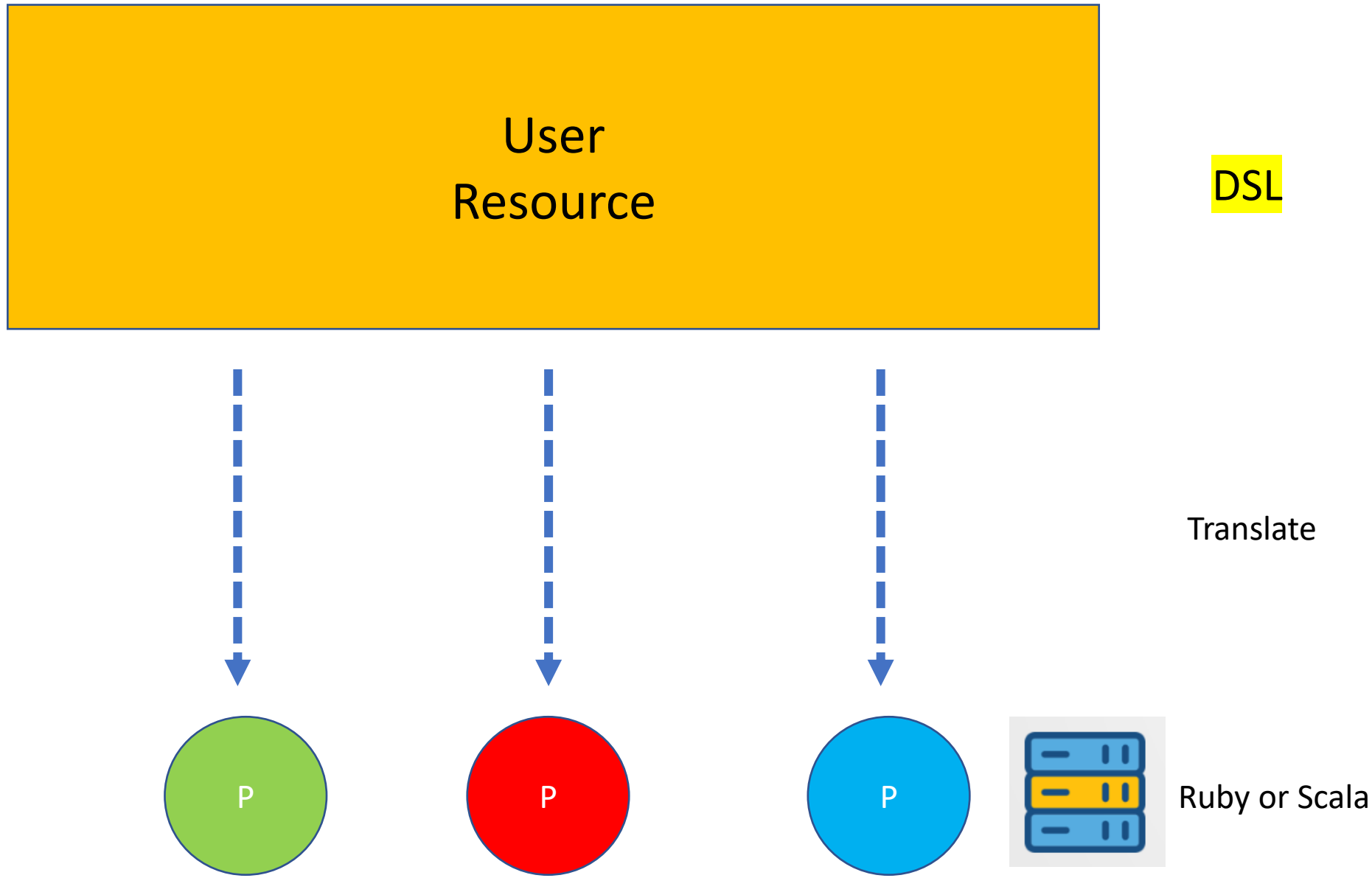
HOW

WHAT

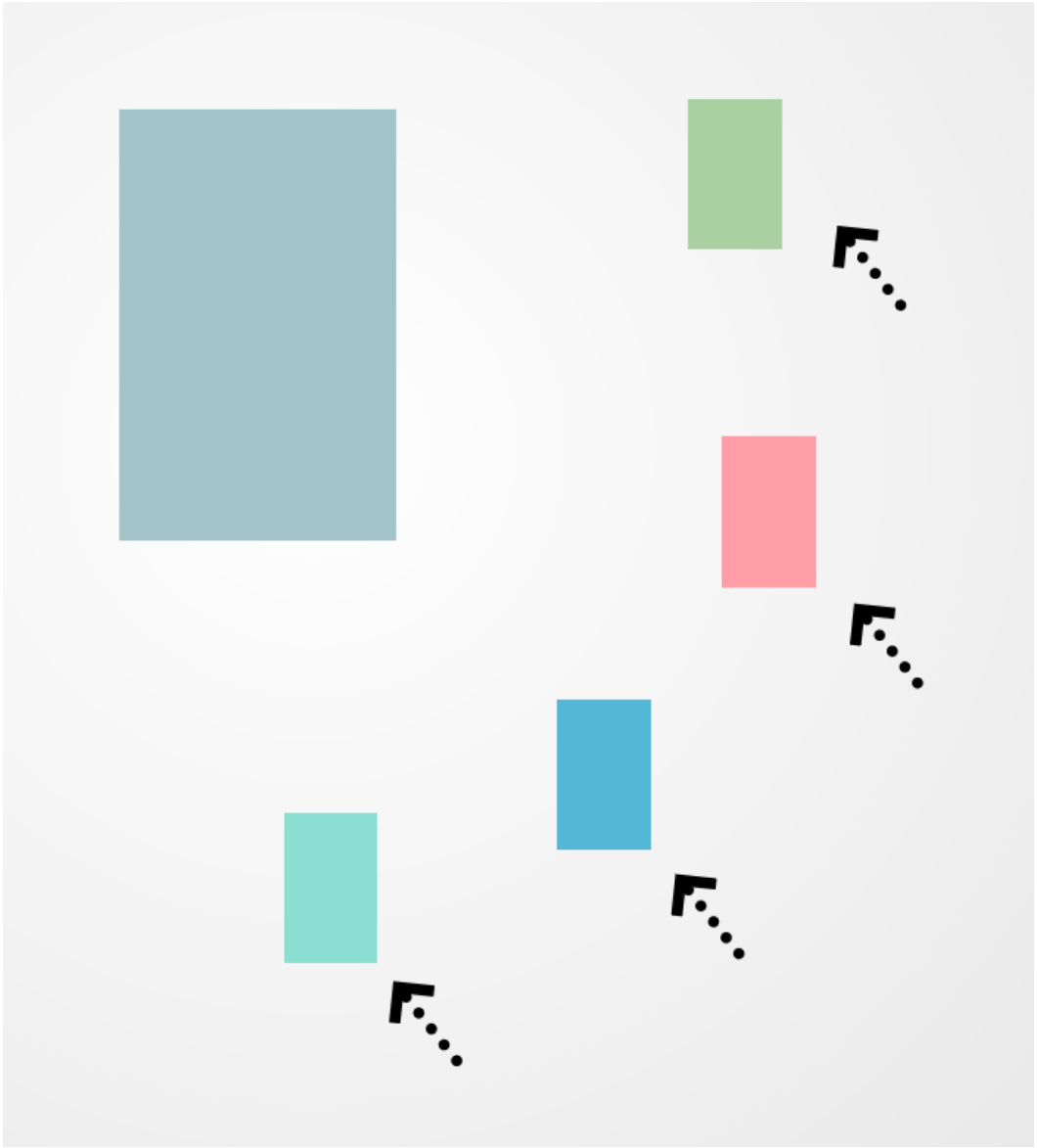
What is IaaC ?

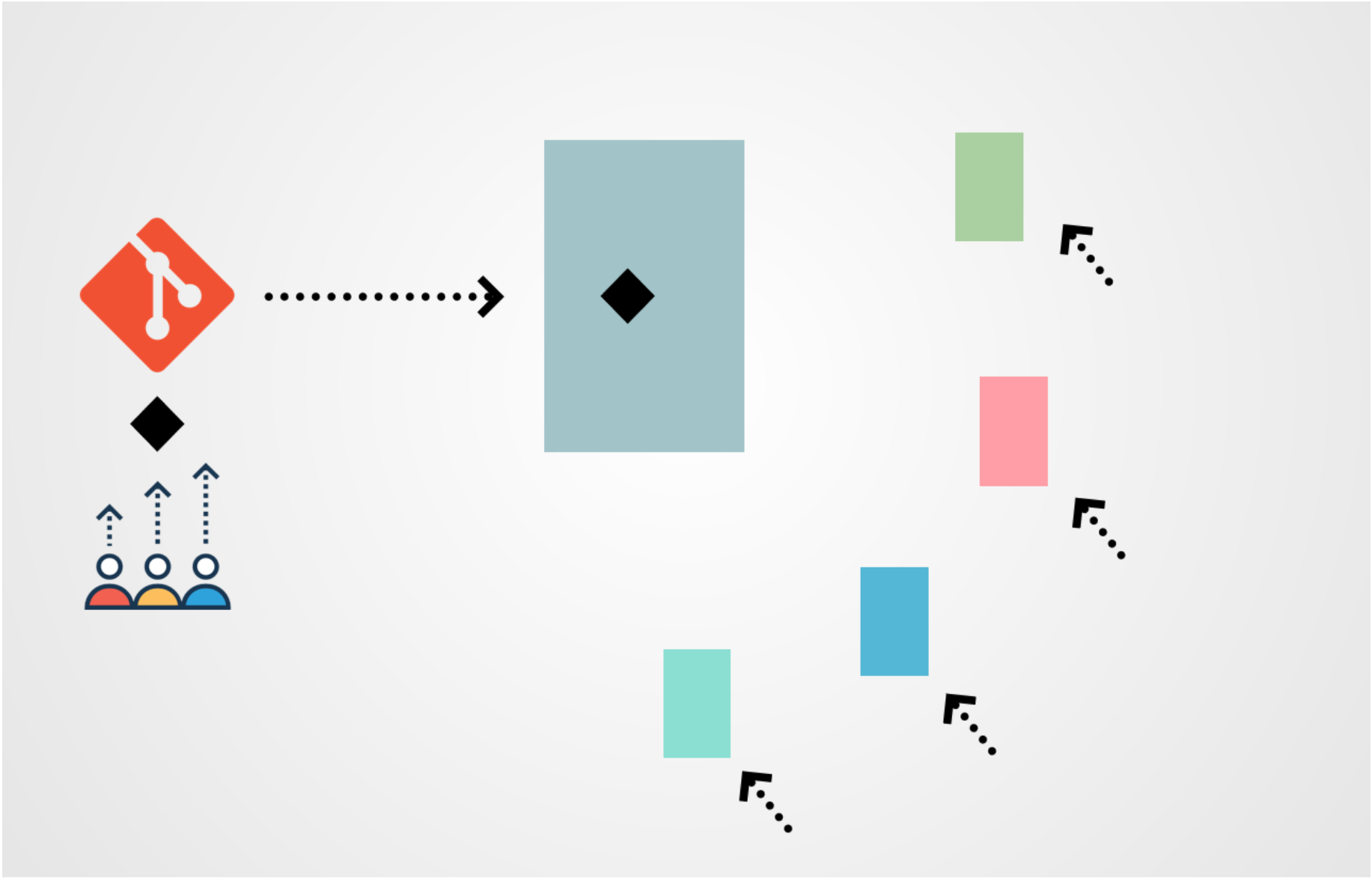
User = gtuser
ID = GT1434
Password = 4813

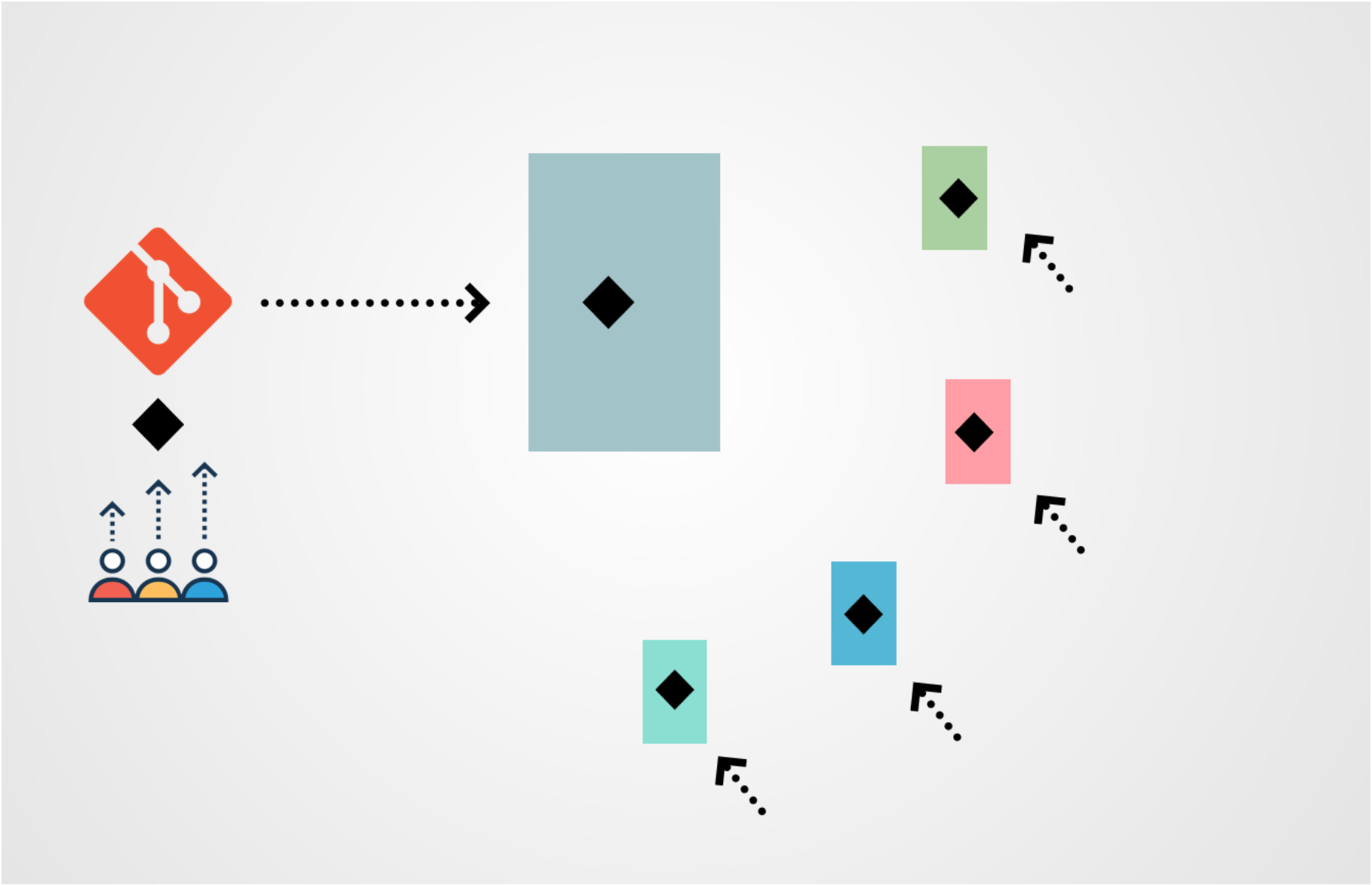
Resource

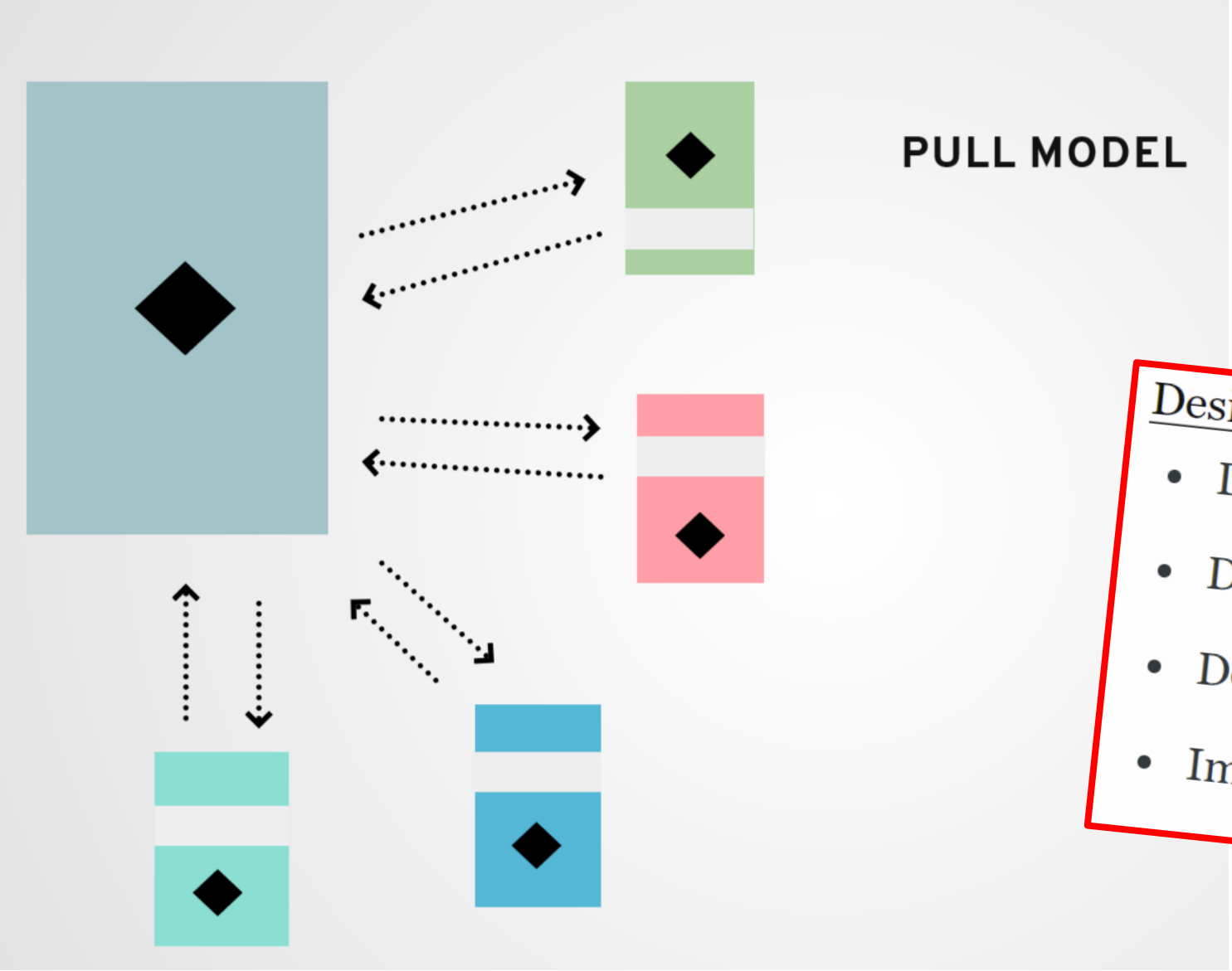

User
Resource
DSL
Translate
P
P
P
Ruby or Scala
Cloud Providers (AWS, Azure, …)

- Declarative Approach “WHAT”
- Independent of Platform
- Revision Control
- Migration
- Consistency



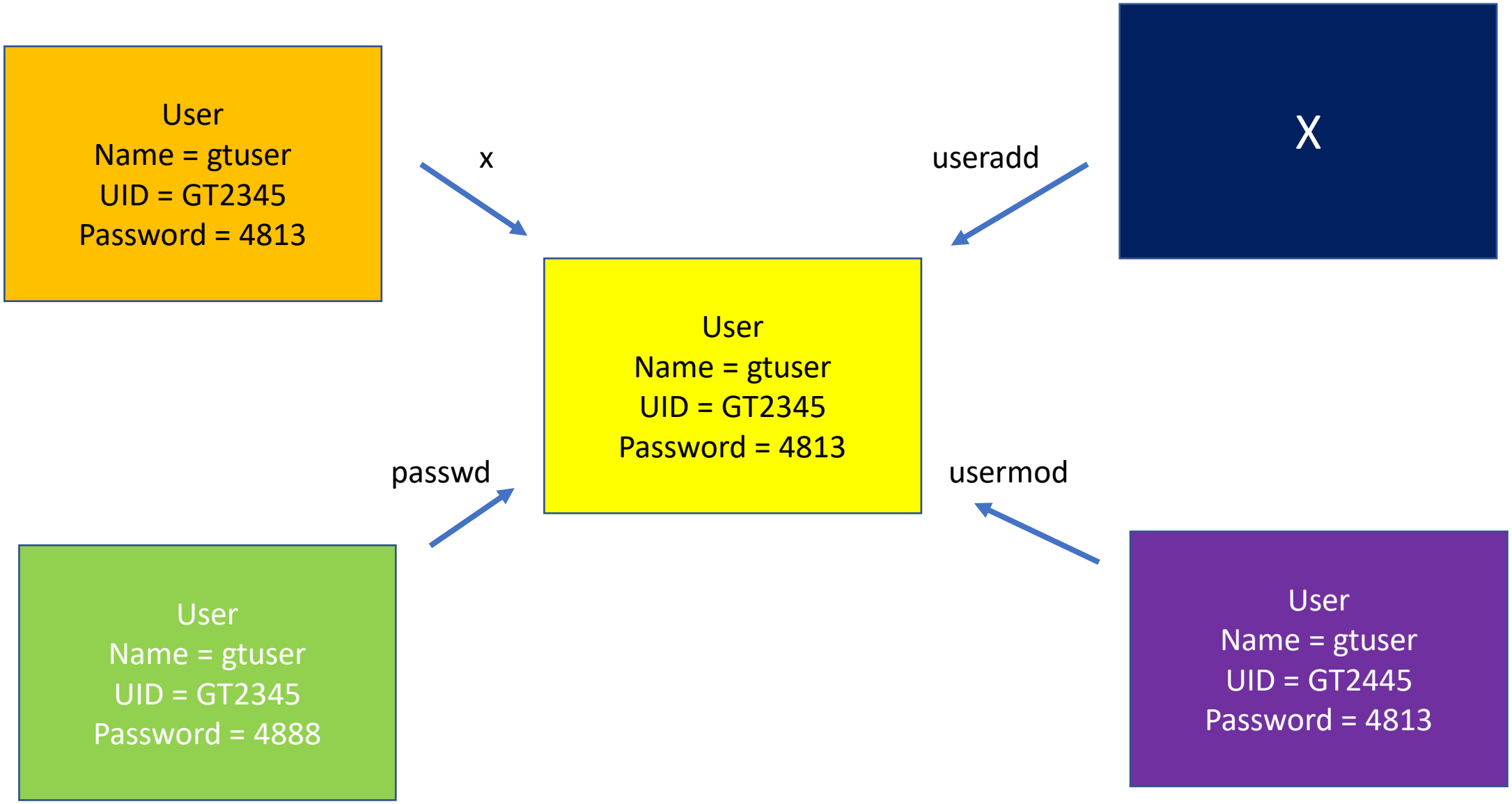






Design principles.

- Declaritive
- Distributed
- Decoupled
- Immutable



Convergent Model
(Focus on the End or Desired State !)

Container Orchestration

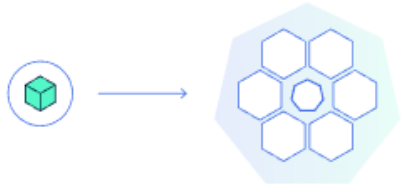
Deploying and scaling containers

Microservices at Netflix

Netflix – Sea of Microservices and their Dependencies

Kubernetes is a production-grade, open-source platform that orchestrates the placement (scheduling) and execution of application containers within and across computer clusters.

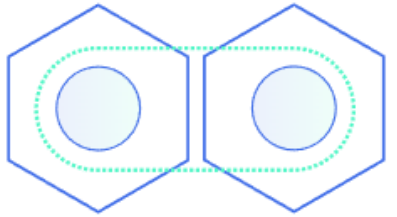
1. Create a Kubernetes cluster



2. Deploy an app

3. Explore your app

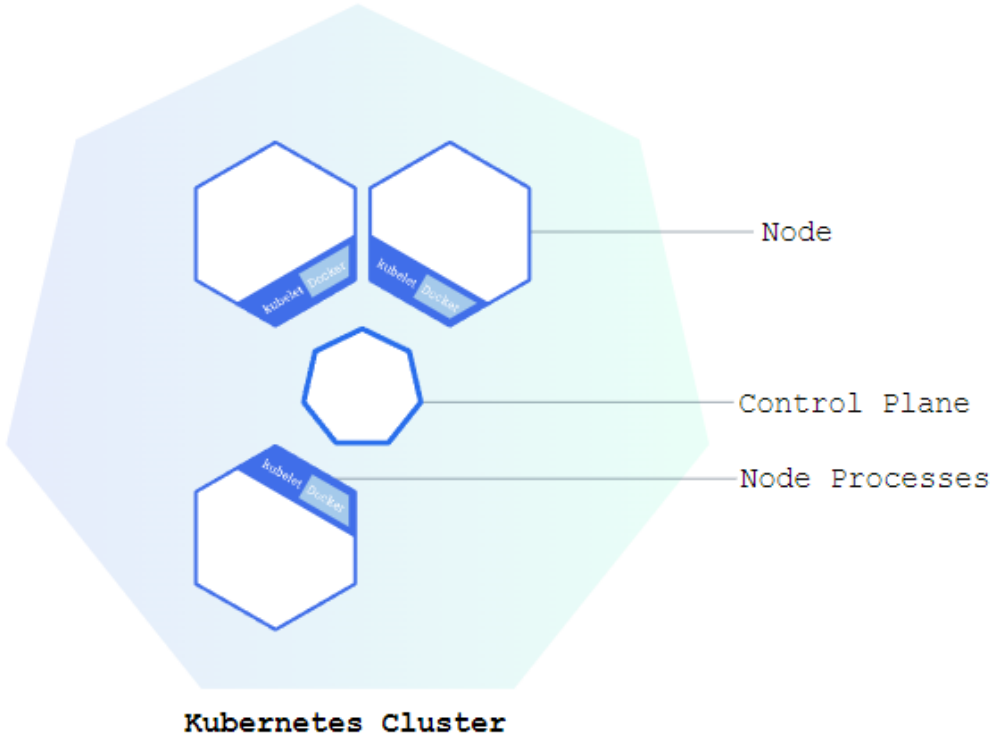
4. Expose your app publicly



5. Scale up your app

6. Update your app

Cluster Diagram



Kubernetes Clusters

Kubernetes coordinates a highly available cluster of computers that are connected to work as a single unit. The abstractions in Kubernetes allow you to deploy containerized applications to a cluster without tying them specifically to individual machines. To make use of this new model of deployment, applications need to be packaged in a way that decouples them from individual hosts: they need to be containerized. Containerized applications are more flexible and available than in past deployment models, where applications were installed directly onto specific machines as packages deeply integrated into the host. **Kubernetes automates the distribution and scheduling of application containers across a cluster in a more efficient way.** Kubernetes is an open-source platform and is production-ready.

A Kubernetes cluster consists of two types of resources:

- The **Control Plane** coordinates the cluster
- **Nodes** are the workers that run applications

```
$ minikube start
* minikube v1.18.0 on Ubuntu 18.04 (amd64)
* Using the none driver based on existing profile

X The requested memory allocation of 2200MiB does not leave room
 for system overhead (total system memory: 2460MiB). You may fac
e stability issues.
* Suggestion: Start minikube with less memory allocated: 'miniku
be start --memory=2200mb'

* Starting control plane node minikube in cluster minikube
* Running on localhost (CPUs=2, Memory=2460MB, Disk=194868MB) ..
.
* OS release is Ubuntu 18.04.5 LTS
* Preparing Kubernetes v1.20.2 on Docker 19.03.13 ...
  - kubelet.resolv-conf=/run/systemd/resolve/resolv.conf
  - Generating certificates and keys ...
  - Booting up control plane ...
  - Configuring RBAC rules ...
* Configuring local host environment ...
* Verifying Kubernetes components...
  - Using image gcr.io/k8s-minikube/storage-provisioner:v4
* Enabled addons: default-storageclass, storage-provisioner
* Done! kubectl is now configured to use "minikube" cluster and
"default" namespace by default
$ date
Sun Jan 29 19:52:32 UTC 2023
$
```

```
* Starting control plane node minikube in cluster minikube
* Running on localhost (CPUs=2, Memory=2460MB, Disk=194868MB) ..
.
* OS release is Ubuntu 18.04.5 LTS
* Preparing Kubernetes v1.20.2 on Docker 19.03.13 ...
  - kubelet.resolv-conf=/run/systemd/resolve/resolv.conf
  - Generating certificates and keys ...
  - Booting up control plane ...
  - Configuring RBAC rules ...
* Configuring local host environment ...
* Verifying Kubernetes components...
  - Using image gcr.io/k8s-minikube/storage-provisioner:v4
* Enabled addons: default-storageclass, storage-provisioner
* Done! kubectl is now configured to use "minikube" cluster and
"default" namespace by default
$ date
Sun Jan 29 19:52:32 UTC 2023
$ kubectl version
Client Version: version.Info{Major:"1", Minor:"20", GitVersion:"
v1.20.4", GitCommit:"e87da0bd6e03ec3fea7933c4b5263d151aafd07c",
GitTreeState:"clean", BuildDate:"2021-02-18T16:12:00Z", GoVersio
n:"go1.15.8", Compiler:"gc", Platform:"linux/amd64"}
Server Version: version.Info{Major:"1", Minor:"20", GitVersion:"
v1.20.2", GitCommit:"faecb196815e248d3ecfb03c680a4507229c2a56",
GitTreeState:"clean", BuildDate:"2021-01-13T13:20:00Z", GoVersio
n:"go1.15.5", Compiler:"gc", Platform:"linux/amd64"}
$
```

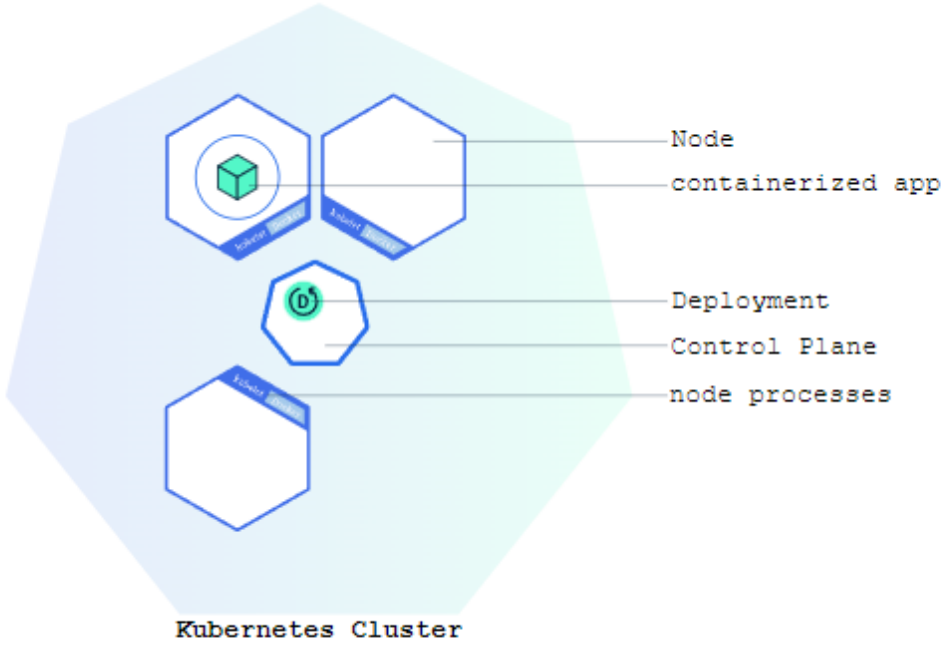
```
$ kubectl version
Client Version: version.Info{Major:"1", Minor:"20", GitVersion:"
v1.20.4", GitCommit:"e87da0bd6e03ec3fea7933c4b5263d151aafd07c",
GitTreeState:"clean", BuildDate:"2021-02-18T16:12:00Z", GoVersio
n:"go1.15.8", Compiler:"gc", Platform:"linux/amd64"}
Server Version: version.Info{Major:"1", Minor:"20", GitVersion:"
v1.20.2", GitCommit:"faecb196815e248d3ecfb03c680a4507229c2a56",
GitTreeState:"clean", BuildDate:"2021-01-13T13:20:00Z", GoVersio
n:"go1.15.5", Compiler:"gc", Platform:"linux/amd64"}
$ kubectl cluster-info
Kubernetes control plane is running at https://10.0.0.14:8443
KubeDNS is running at https://10.0.0.14:8443/api/v1/namespaces/k
ube-system/services/kube-dns:dns/proxy

To further debug and diagnose cluster problems, use 'kubectl clu
ster-info dump'.
$ kuberctl get nodes

Command 'kuberctl' not found, did you mean:

  command 'kubectl' from snap kubectl (1.26.1)

See 'snap info <snapname>' for additional versions.

$ kubectl get nodes
NAME       STATUS   ROLES                  AGE     VERSION
minikube   Ready    control-plane,master   3m28s   v1.20.2
$
```

Kubernetes Deployments

Once you have a running Kubernetes cluster, you can deploy your containerized applications on top of it. To do so, you create a Kubernetes **Deployment** configuration. The Deployment instructs Kubernetes how to create and update instances of your application. Once you've created a Deployment, the Kubernetes control plane schedules the application instances included in that Deployment to run on individual Nodes in the cluster.

Once the application instances are created, a Kubernetes Deployment Controller continuously monitors those instances. If the Node hosting an instance goes down or is deleted, the Deployment controller replaces the instance with an instance on another Node in the cluster. **This provides a self-healing mechanism to address machine failure or maintenance.**

In a pre-orchestration world, installation scripts would often be used to start applications, but they did not allow recovery from machine failure. By both creating your application instances and keeping them running across Nodes, Kubernetes Deployments provide a fundamentally different approach to application management.



A Pod is a group of one or more application containers (such as Docker) and includes shared storage (volumes), IP address and information about how to run them.

Kubernetes Pods

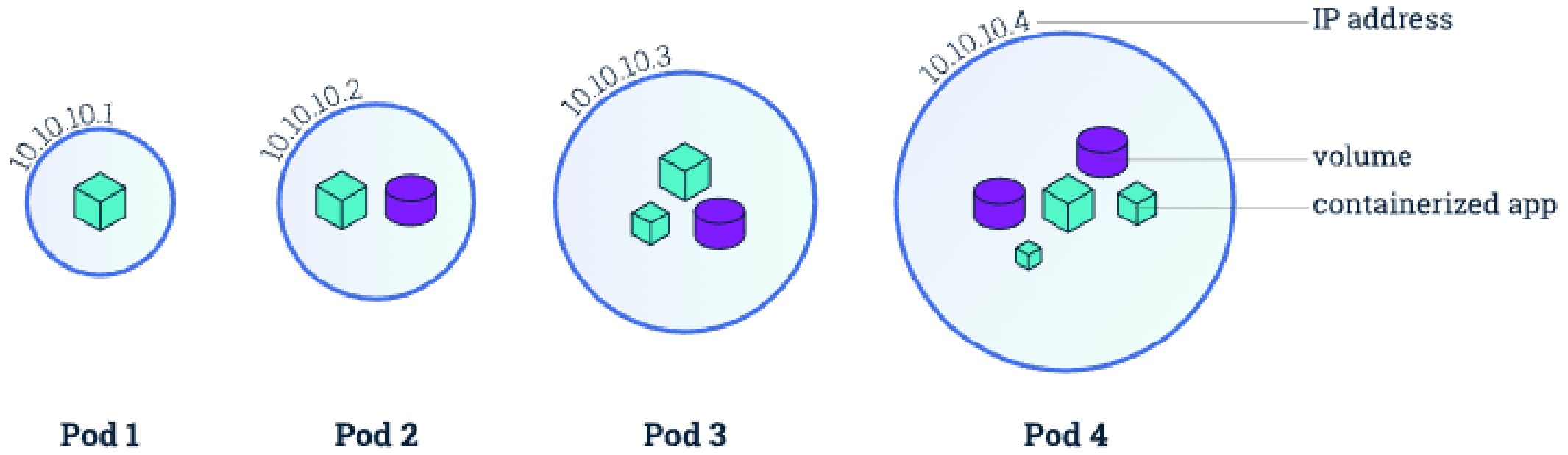
When you created a Deployment in Module 2, Kubernetes created a **Pod** to host your application instance. A Pod is a Kubernetes abstraction that represents a group of one or more application containers (such as Docker), and some shared resources for those containers. Those resources include:

- Shared storage, as Volumes
- Networking, as a unique cluster IP address
- Information about how to run each container, such as the container image version or specific ports to use

A Pod models an application-specific "logical host" and can contain different application containers which are relatively tightly coupled. For example, a Pod might include both the container with your Node.js app as well as a different container that feeds the data to be published by the Node.js webserver. The containers in a Pod share an IP Address and port space, are always co-located and co-scheduled, and run in a shared context on the same Node.

Pods are the atomic unit on the Kubernetes platform. When we create a Deployment on Kubernetes, that Deployment creates Pods with containers inside them (as opposed to creating containers directly). Each Pod is tied to the Node where it is scheduled, and remains there until termination (according to restart policy) or deletion. In case of a Node failure, identical Pods are scheduled on other available Nodes in the cluster.

Pods overview



Nodes

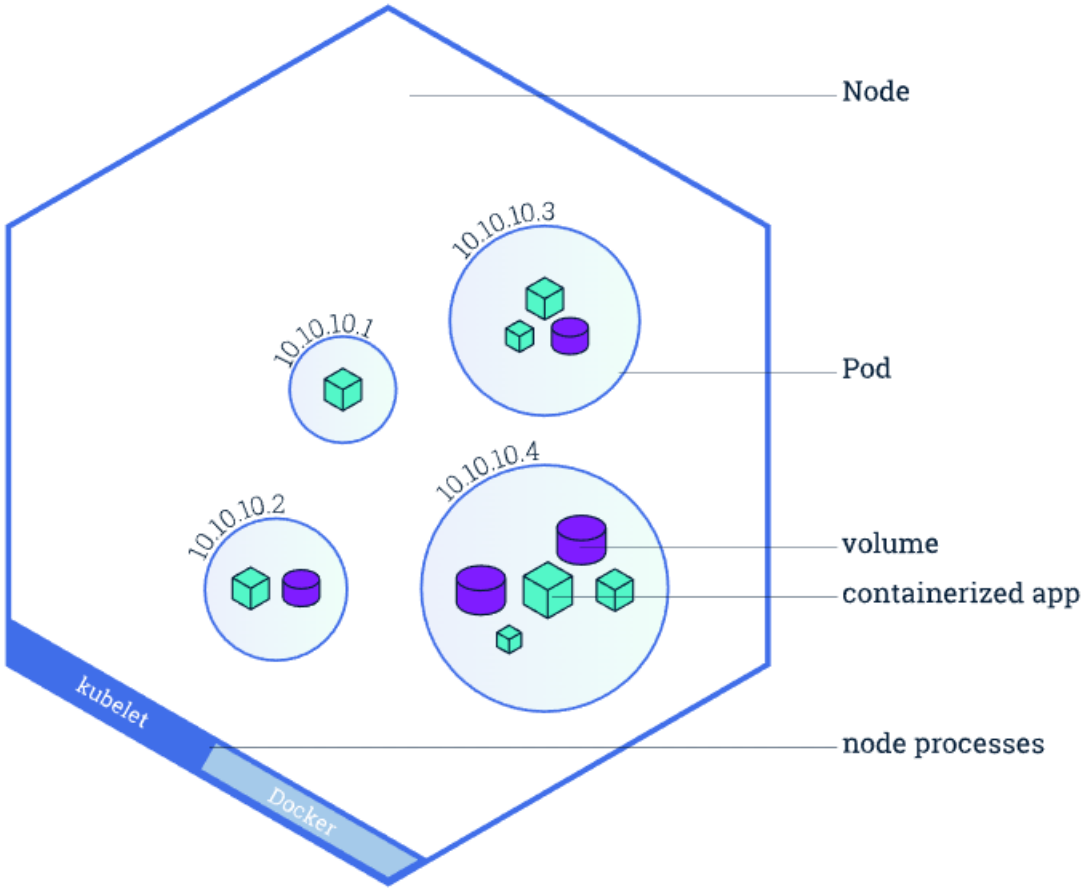
A Pod always runs on a **Node**. A Node is a worker machine in Kubernetes and may be either a virtual or a physical machine, depending on the cluster. Each Node is managed by the control plane. A Node can have multiple pods, and the Kubernetes control plane automatically handles scheduling the pods across the Nodes in the cluster. The control plane's automatic scheduling takes into account the available resources on each Node.

Every Kubernetes Node runs at least:

- Kubelet, a process responsible for communication between the Kubernetes control plane and the Node; it manages the Pods and the containers running on a machine.
- A container runtime (like Docker) responsible for pulling the container image from a registry, unpacking the container, and running the application.

Containers should only be scheduled together in a single Pod if they are tightly coupled and need to share resources such as disk.

Node overview



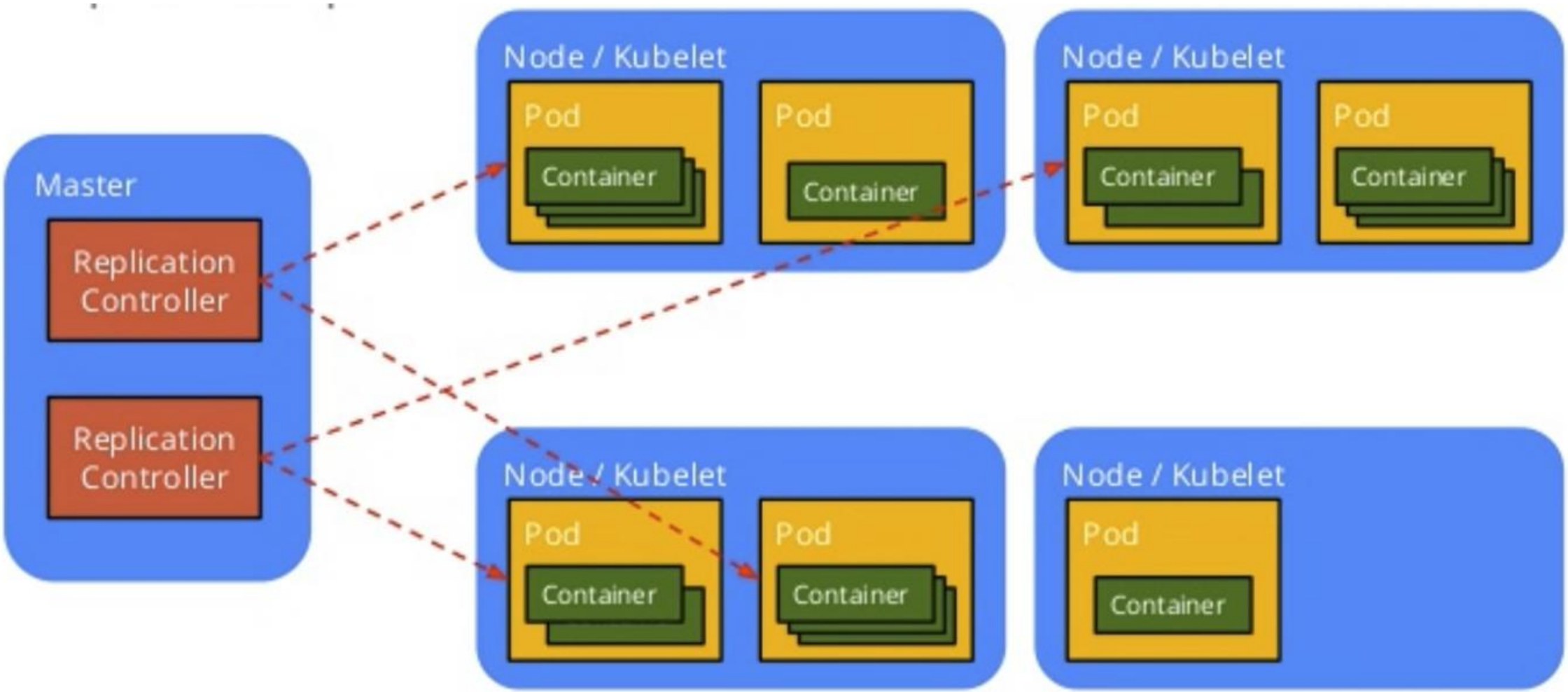
Terminal +

```
$ date
Sun Jan 29 20:03:52 UTC 2023
$ kubectl get pods
NAME                                   READY   STATUS    RESTART
kubernetes-bootcamp-fb5c67579-w7bw7    0/1     Pending   0
$ kubectl describe pods
Name:         kubernetes-bootcamp-fb5c67579-w7bw7
Namespace:    default
Priority:     0
Node:         minikube/10.0.0.24
Start Time:   Sun, 29 Jan 2023 20:04:03 +0000
Labels:       app=kubernetes-bootcamp
              pod-template-hash=fb5c67579
Annotations:  <none>
Status:       Running
IP:           172.18.0.6
IPs:
  IP:           172.18.0.6
Controlled By:  ReplicaSet/kubernetes-bootcamp-fb5c67579
Containers:
  kubernetes-bootcamp:
    Container ID:   docker://510987d2d231b22a4b44926c9b6e9ecdb
2a2394ba58c9
    Image:          gcr.io/google-samples/kubernetes-bootcamp:
    Image ID:       docker-pullable://jocatalin/kubernetes-boo
e63bb57c5f5b6156f446b3bc3b3c143d233037f3a2f00e279c8fcc64af
    Port:           8080/TCP
    Host Port:      0/TCP
    State:          Running
      Started:      Sun, 29 Jan 2023 20:04:06 +0000
    Ready:          True
    Restart Count:  0
```

```
ts:
ar/run/secrets/kubernetes.io/serviceaccount from def
ns:
                Status
lized           True
                True
nersReady       True
eduled          True

t-token-6m9r4:
:           Secret (a volume populated by a Secret)
etName:     default-token-6m9r4
onal:       false
s:          BestEffort
ectors:     <none>
ons:        node.kubernetes.io/not-ready:NoExecute op=E
            node.kubernetes.io/unreachable:NoExecute op

  Reason      Age   From               Message
  ------      ----  ----               -------
  Scheduled   8s    default-scheduler  Successfully a
otcamp-fb5c67579-w7bw7 to minikube
  Pulled      6s    kubelet            Container imag
rnetes-bootcamp:v1" already present on machine
  Created     5s    kubelet            Created contai

  Started     5s    kubelet            Started contai
```

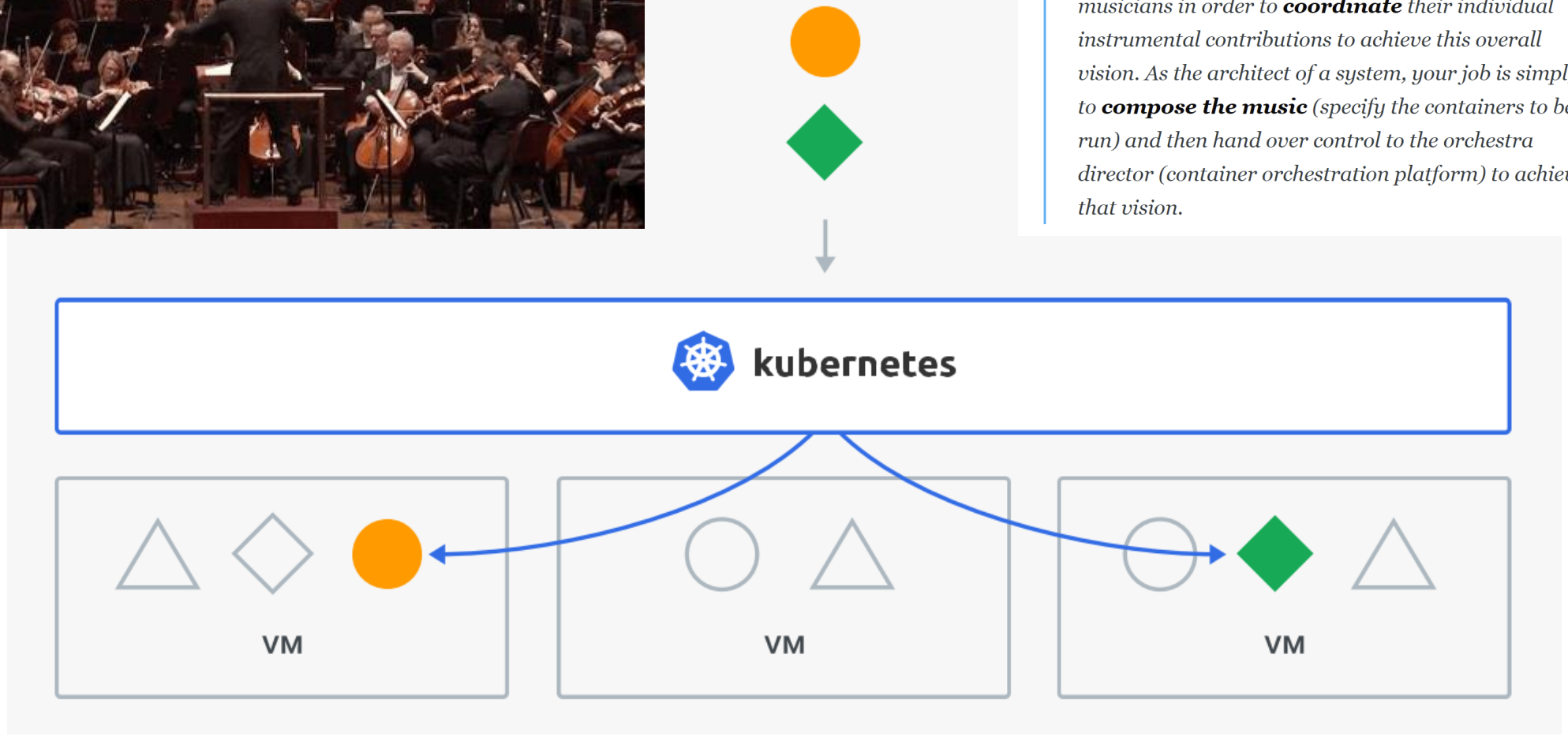
Kubernetes Vocab

- Node
  - Kubelet
  - Communicates with master
  - Runs pods
- Pod
  - Runs 1+ containers
  - Exists on a node
- Service
  - Handles requests
  - Usually a load balancer
- Deployment
  - Defines desired state - kubernetes handles the rest

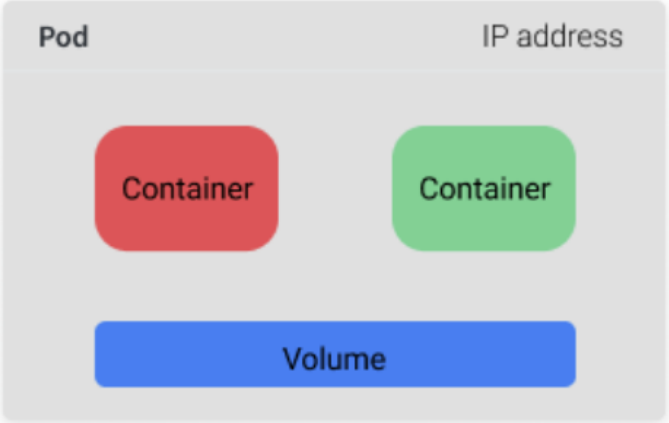

Master
Replication Controller
Replication Controller
Node / Kubelet
Pod
Container
Pod
Container
Node / Kubelet
Pod
Container
Pod
Container
Node / Kubelet
Pod
Container
Pod
Container
Node / Kubelet
Pod
Container

Kubernetes as a container orchestration platform. We can address many of the concerns mentioned above using a container orchestration platform.

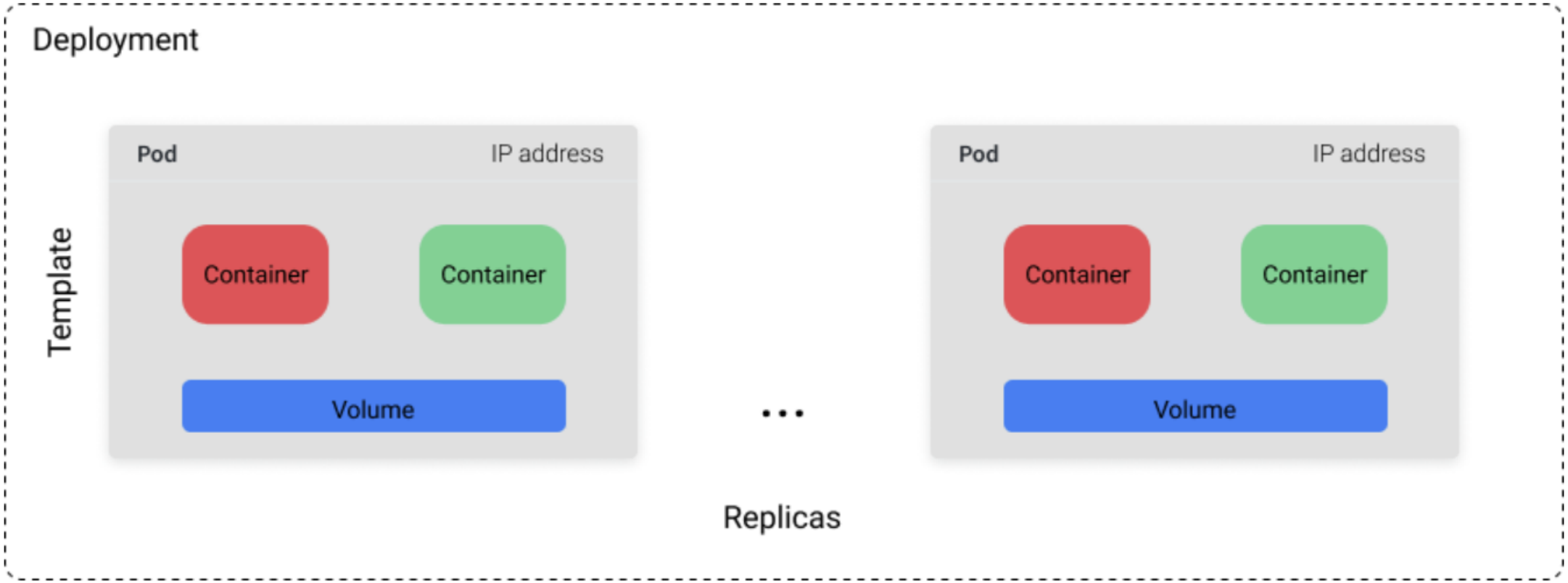
> *The director of an orchestra holds the **vision** for a musical performance and **communicates** with the musicians in order to **coordinate** their individual instrumental contributions to achieve this overall vision. As the architect of a system, your job is simply to **compose the music** (specify the containers to be run) and then hand over control to the orchestra director (container orchestration platform) to achieve that vision.*

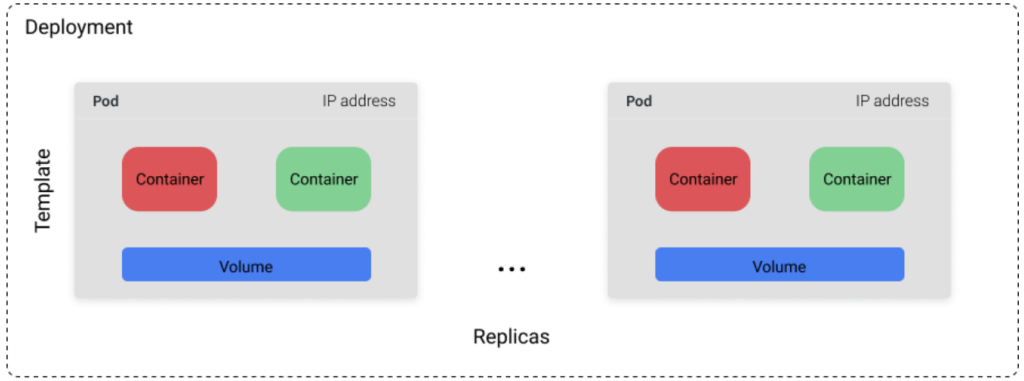


The **Pod** object is the fundamental building block in Kubernetes, comprised of one or more (tightly related) containers, a shared networking layer, and shared filesystem volumes. Similar to containers, pods are designed to be ephemeral - there is no expectation that a *specific, individual pod* will persist for a long lifetime.



https://www.jeremyjordan.me/ml-monitoring/


Deployment
Template
Pod
IP address
Container
Container
Volume
Pod
IP address
Container
Container
Volume
Replicas




```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: ml-model-serving
  labels:
    app: ml-model
spec:
  replicas: 10
  selector:
    matchLabels:
      app: ml-model
  template:
    metadata:
      labels:
        app: ml-model
    spec:
      containers:
      - name: ml-rest-server
        image: ml-serving:1.0
        ports:
        - containerPort: 80
```

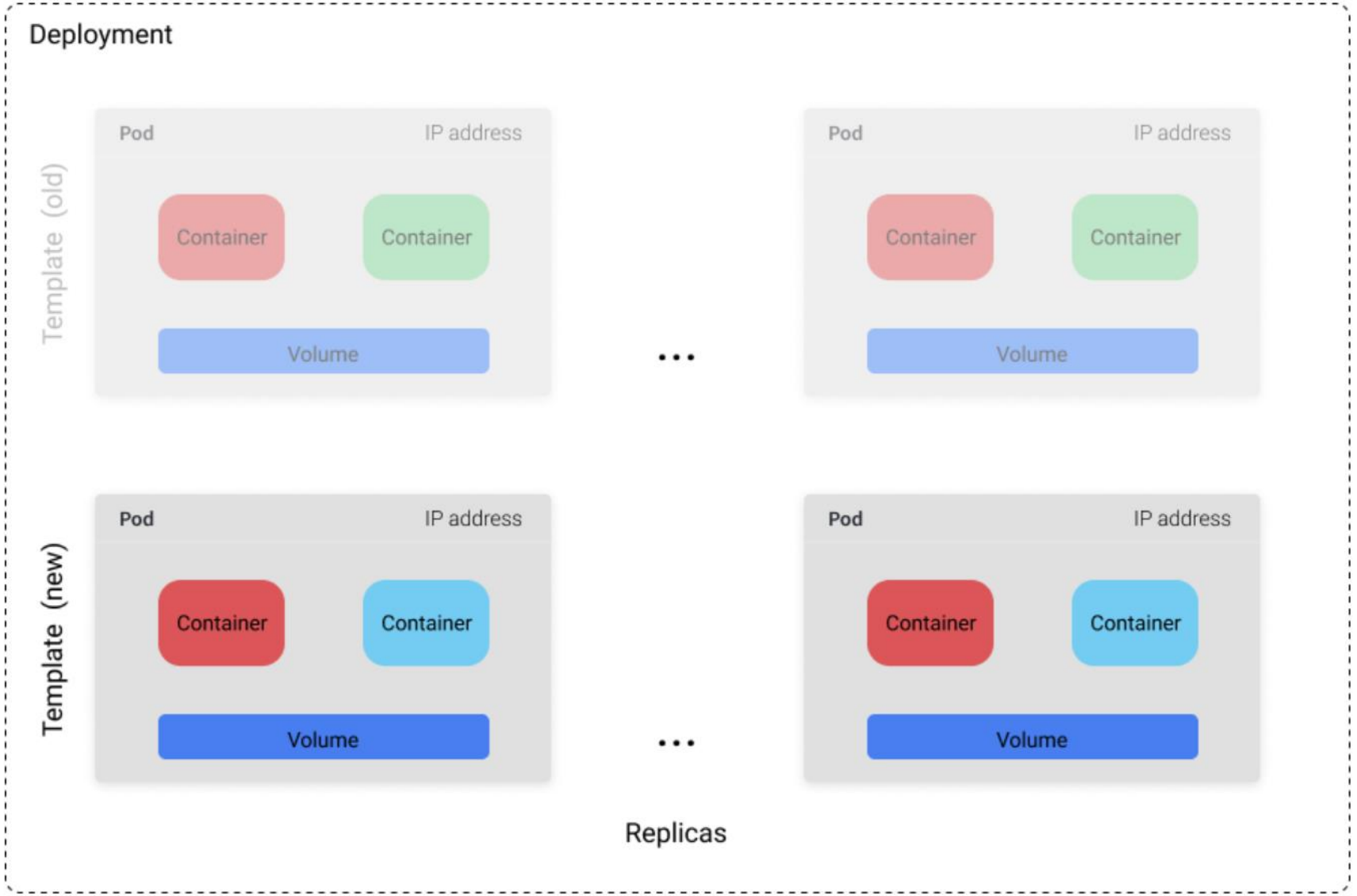
How many Pods should be running?

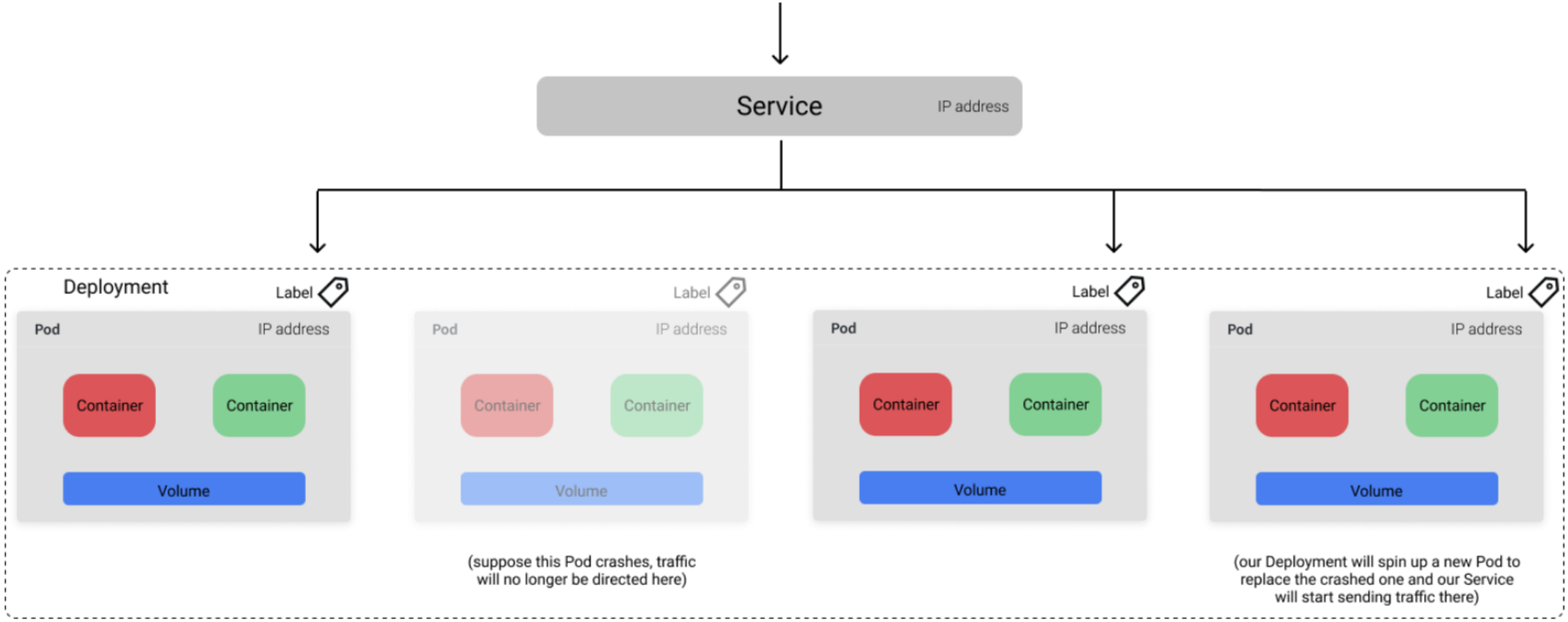
How do we find Pods that belong to this Deployment?

What should a Pod look like?

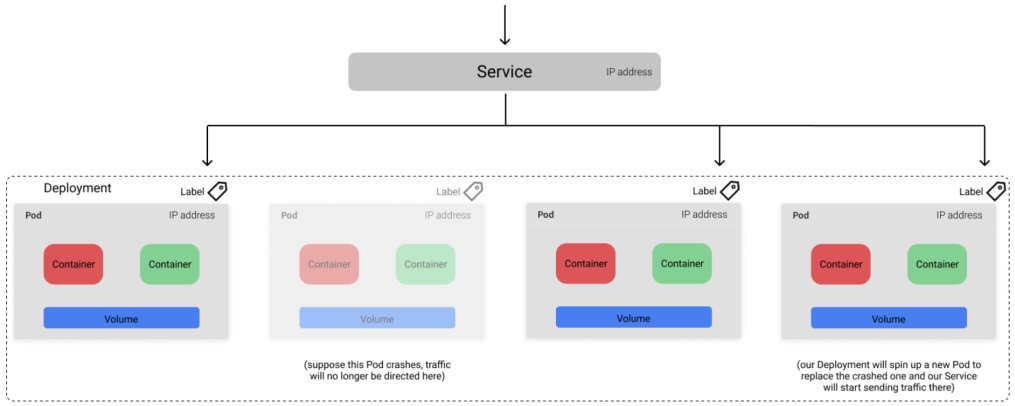
Add a label to the Pods so our Deployment can find the Pods to manage.

What containers should be running in the Pod?


Deployment
Template (old)
Pod
IP address
Container
Container
Volume
Pod
IP address
Container
Container
Volume
Template (new)
Pod
IP address
Container
Container
Volume
Pod
IP address
Container
Container
Volume
Replicas



In this example, our Service sends traffic to all healthy Pods with the label `app="ml-model"`.



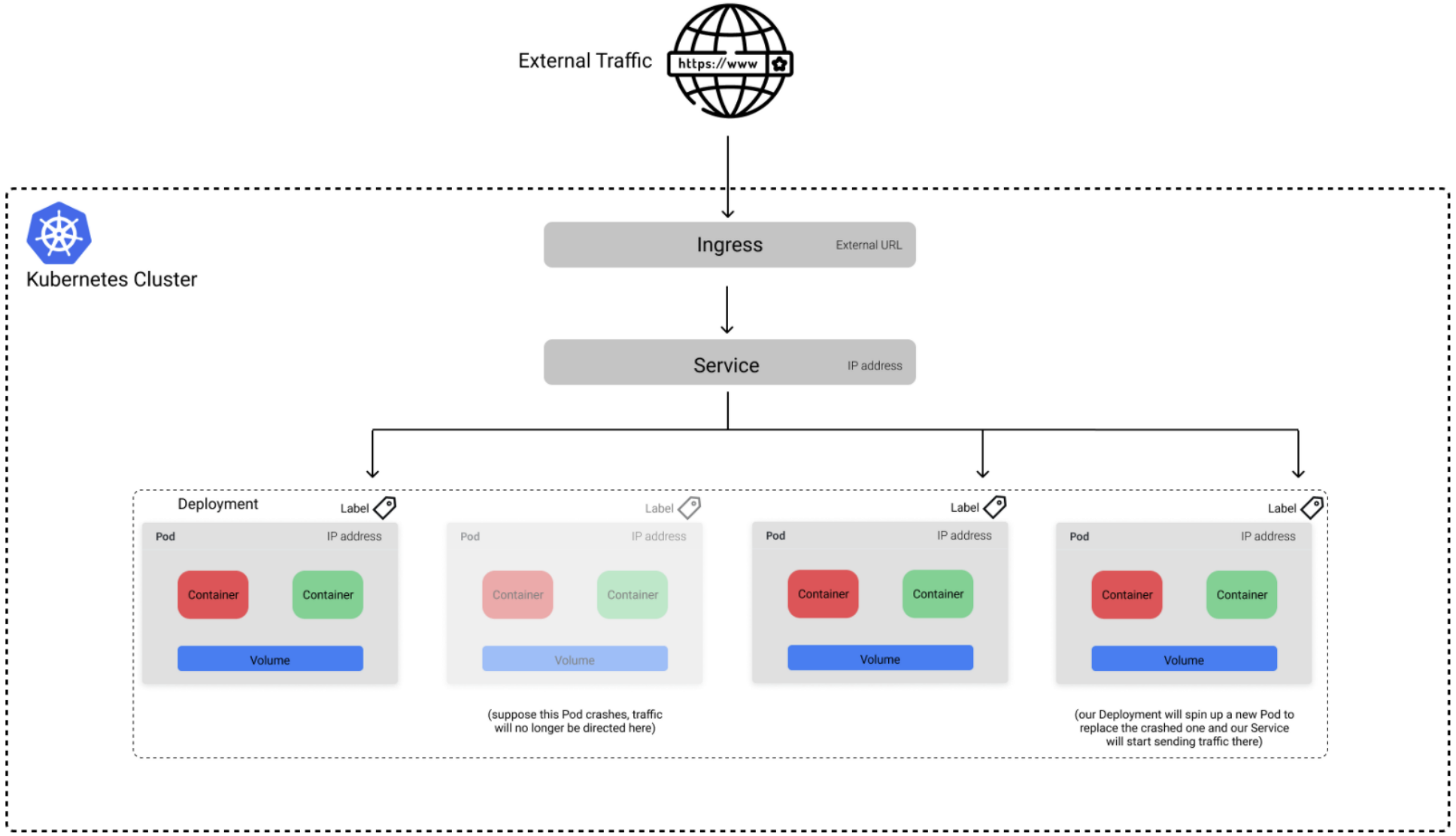
In this example, our Service sends traffic to all healthy Pods with the label `app="ml-model"`.

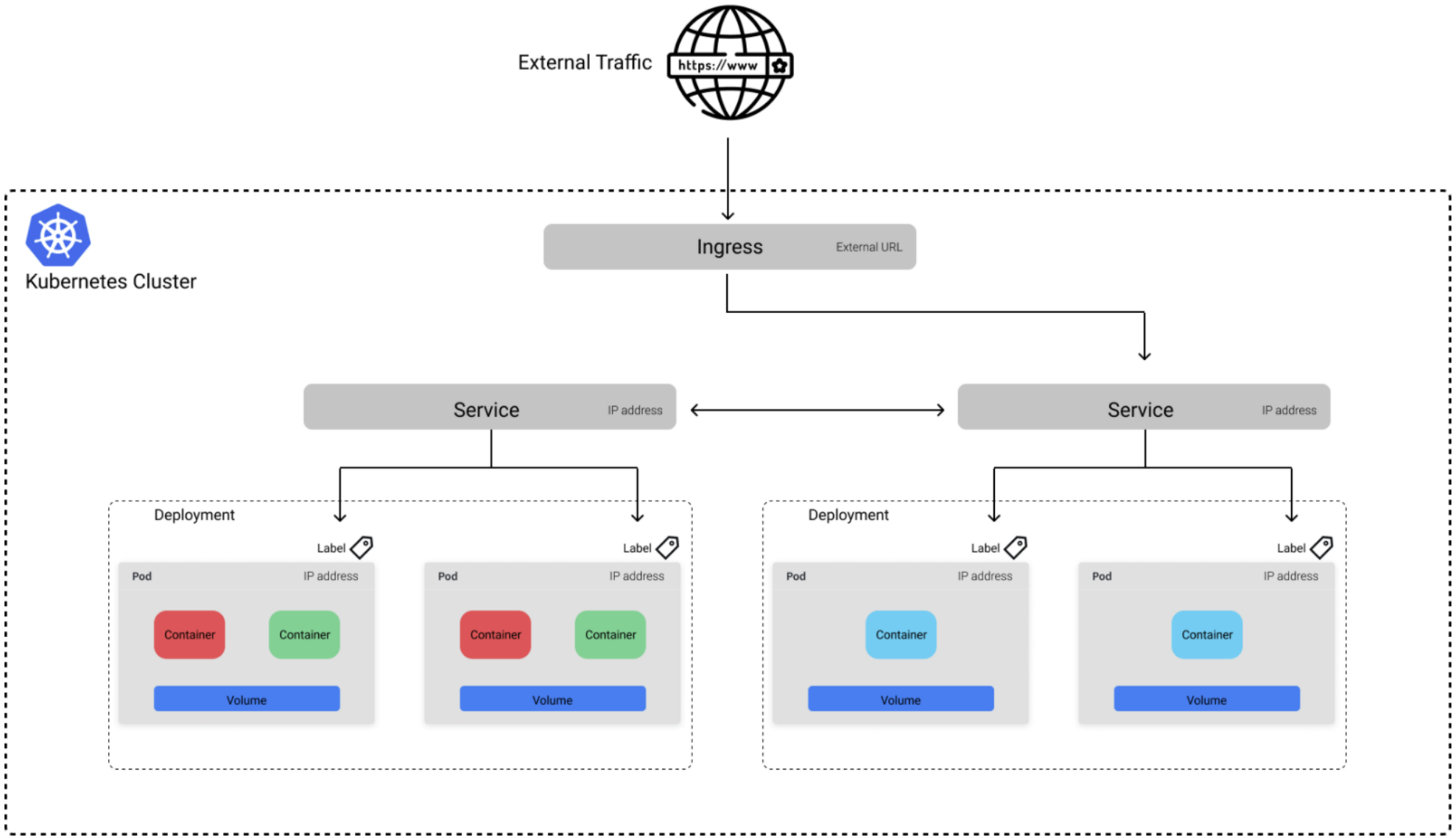
```yaml
apiVersion: v1
kind: Service
metadata:
  name: ml-model-svc
  labels:
    app: ml-model
spec:
  type: ClusterIP
  selector:
    app: ml-model
  ports:
  - protocol: TCP
    port: 80
```

How do we want to expose our endpoint?

How do we find Pods to direct traffic to?

How will clients talk to our Service?


External Traffic
https://www
Kubernetes Cluster
Ingress
External URL
Service
IP address
Deployment
Label
Pod
IP address
Container
Container
Volume
(suppose this Pod crashes, traffic will no longer be directed here)
(our Deployment will spin up a new Pod to replace the crashed one and our Service will start sending traffic there)

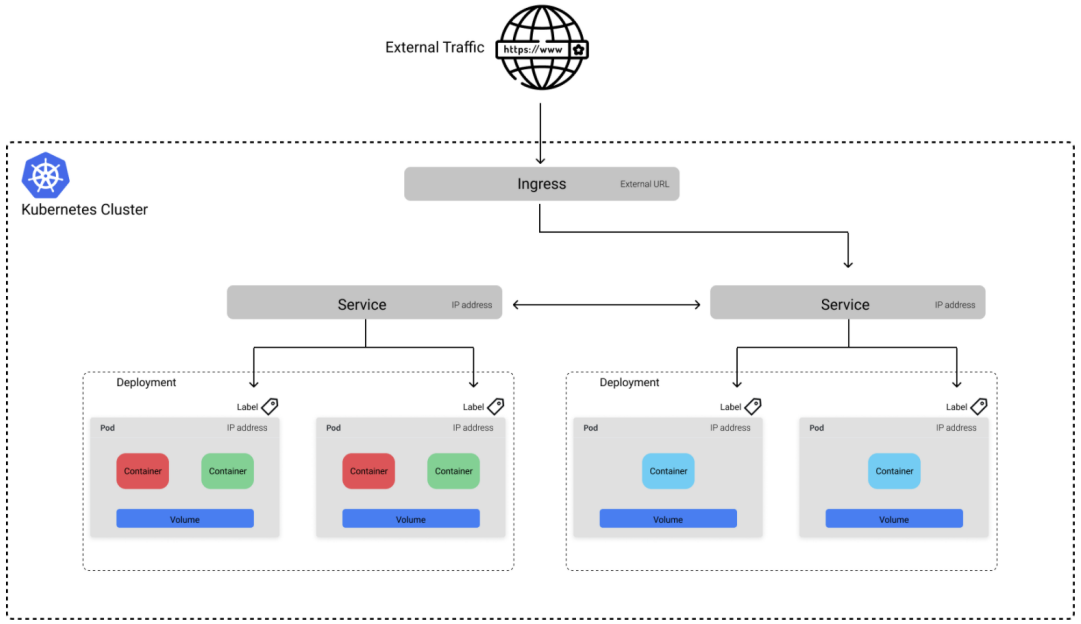

External Traffic
https://www
Kubernetes Cluster
Ingress
External URL
Service
IP address
Service
IP address
Deployment
Label
Label
Pod
IP address
Pod
IP address
Container
Container
Container
Container
Volume
Volume
Deployment
Label
Label
Pod
IP address
Pod
IP address
Container
Container
Volume
Volume

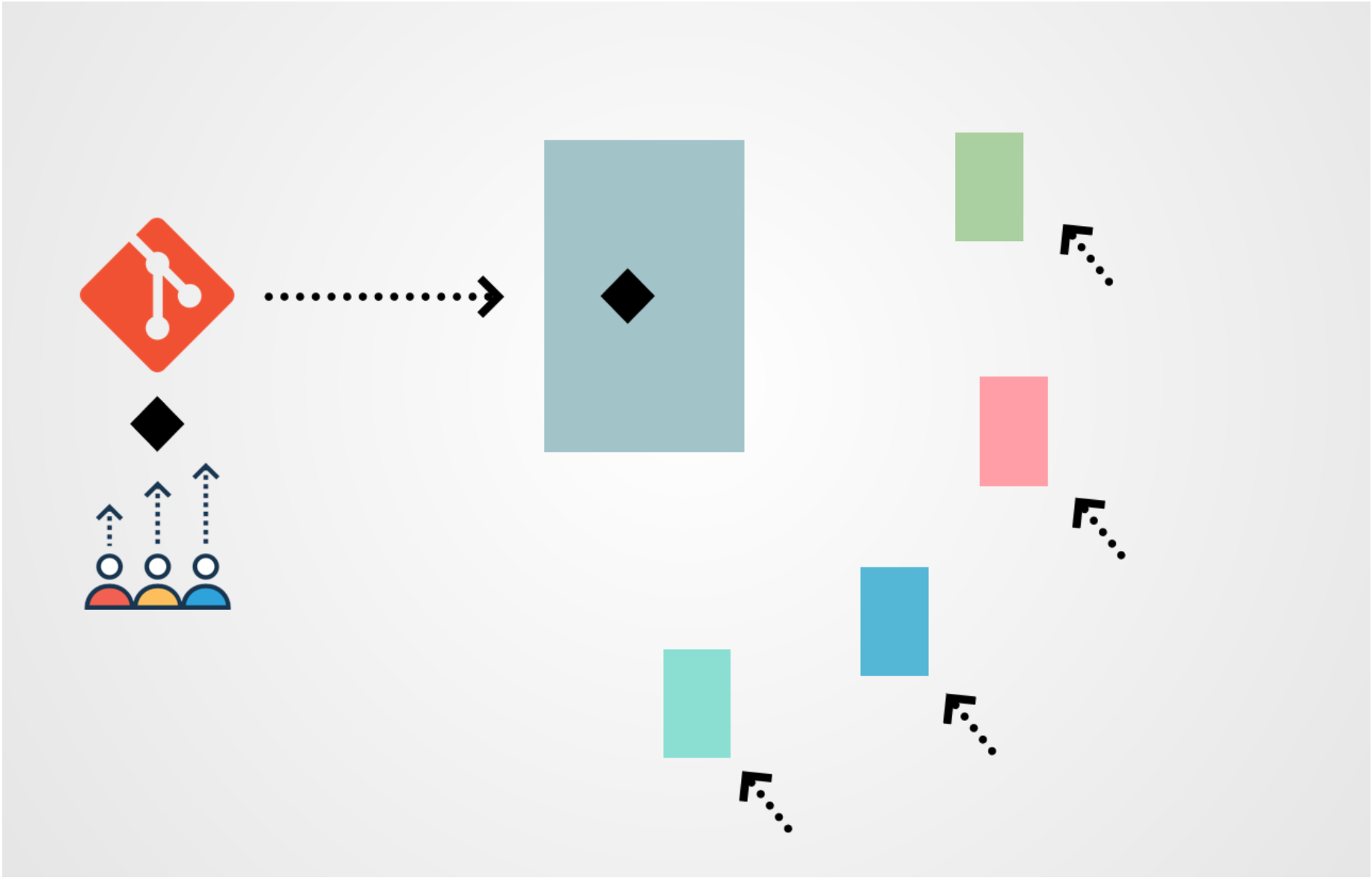
```yaml
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
  name: ml-product-ingress
  annotations:
    kubernetes.io/ingress.class: "nginx"
    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  rules:
  - http:
      paths:
      - path: /app
        backend:
          serviceName: user-interface-svc
          servicePort: 80
```

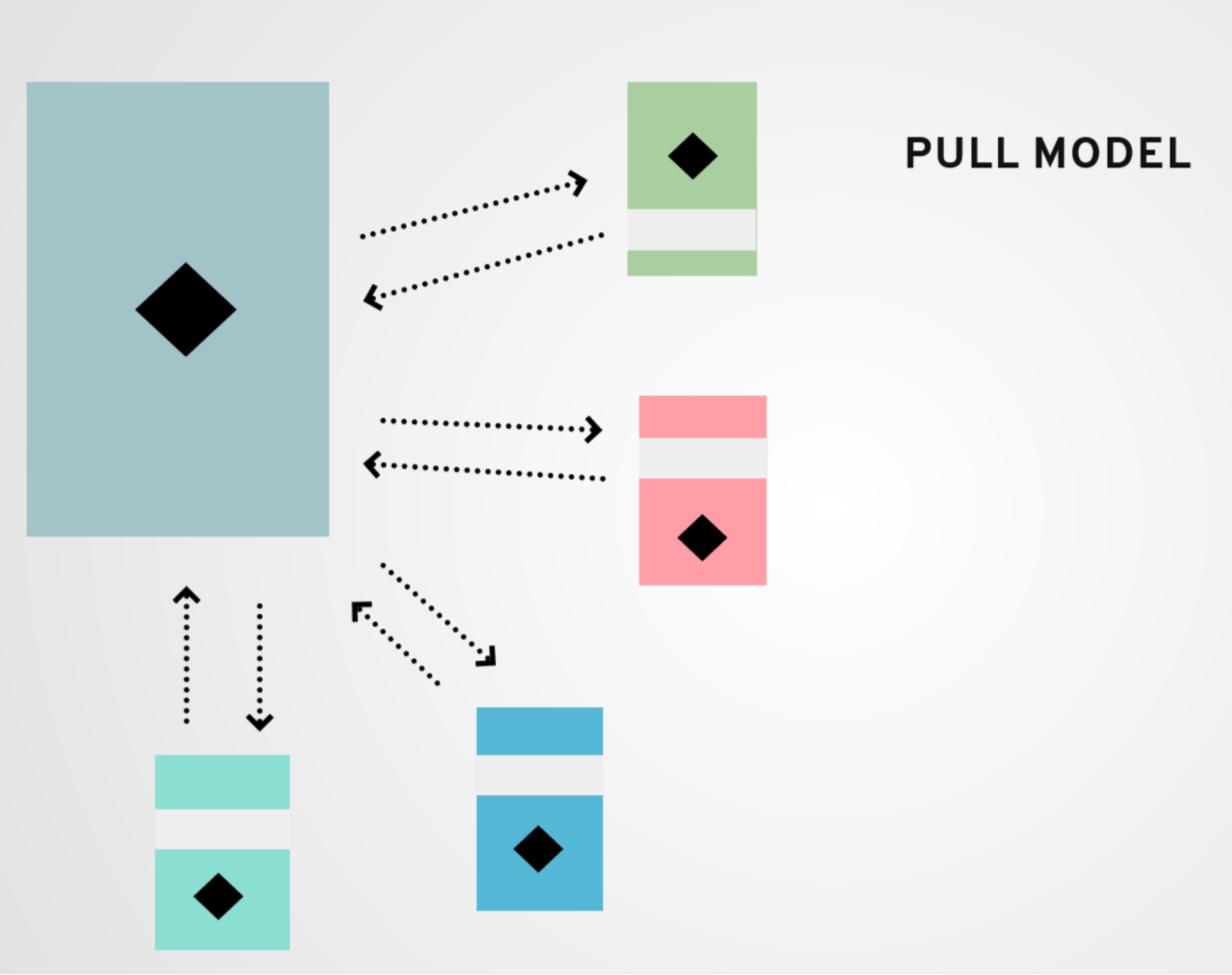
Configure options for the Ingress controller.

How should external traffic access the service?

What Service should we direct traffic to?






PULL MODEL

What is Kubernetes?

- A **Production-Grade Container Orchestration System** Google-grown, based on Borg and Omega, systems that run inside of Google right now and are proven to work at Google for over 10 years.
- Google spawns billions of containers per week with these systems.
- Created by three Google employees initially during the summer of 2014; grew exponentially and became the first project to get donated to the CNCF.
- Hit the first production-grade version v1.0.1 in July 2015. Has continually released a new minor version every three months since v1.2.0 in March 2016. Lately v1.13.0 was released in December 2018.

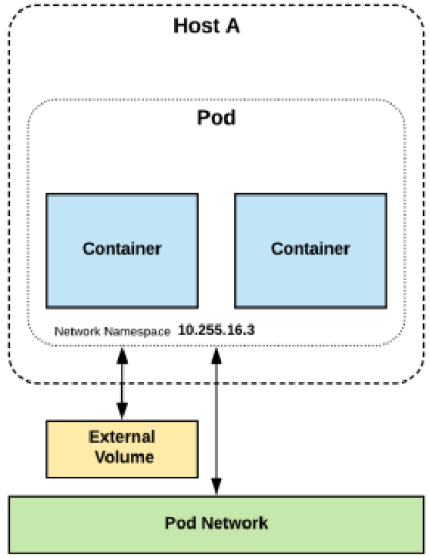
Self-Healing

Kubernetes will **ALWAYS** try and steer the cluster to its desired state.

- **Me:** “I want 3 healthy instances of redis to always be running.”
- **Kubernetes:** “Okay, I’ll ensure there are always 3 instances up and running.”
- **Kubernetes:** “Oh look, one has died. I’m going to attempt to spin up a new one.”

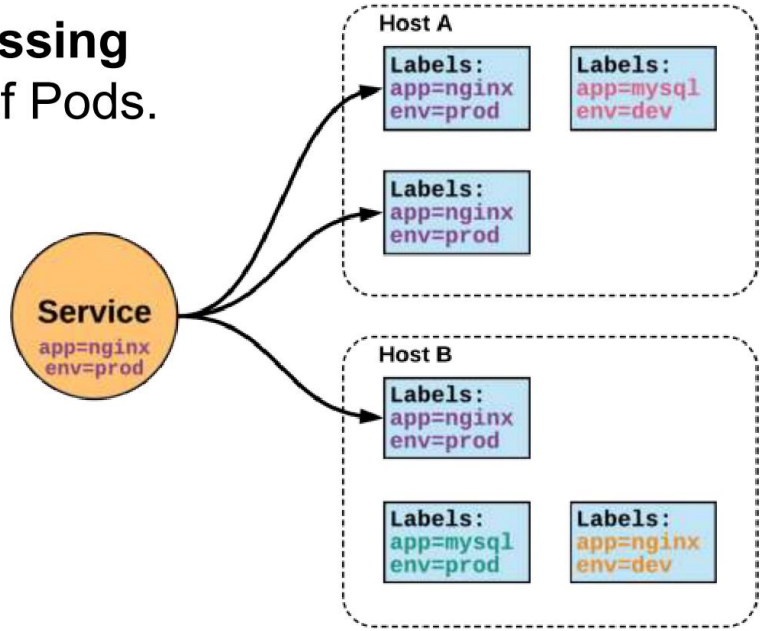
Pods

- **Atomic unit** or smallest "*unit of work*"of Kubernetes.
- Pods are **one or MORE containers** that share volumes and namespace.
- **They are also ephemeral!**

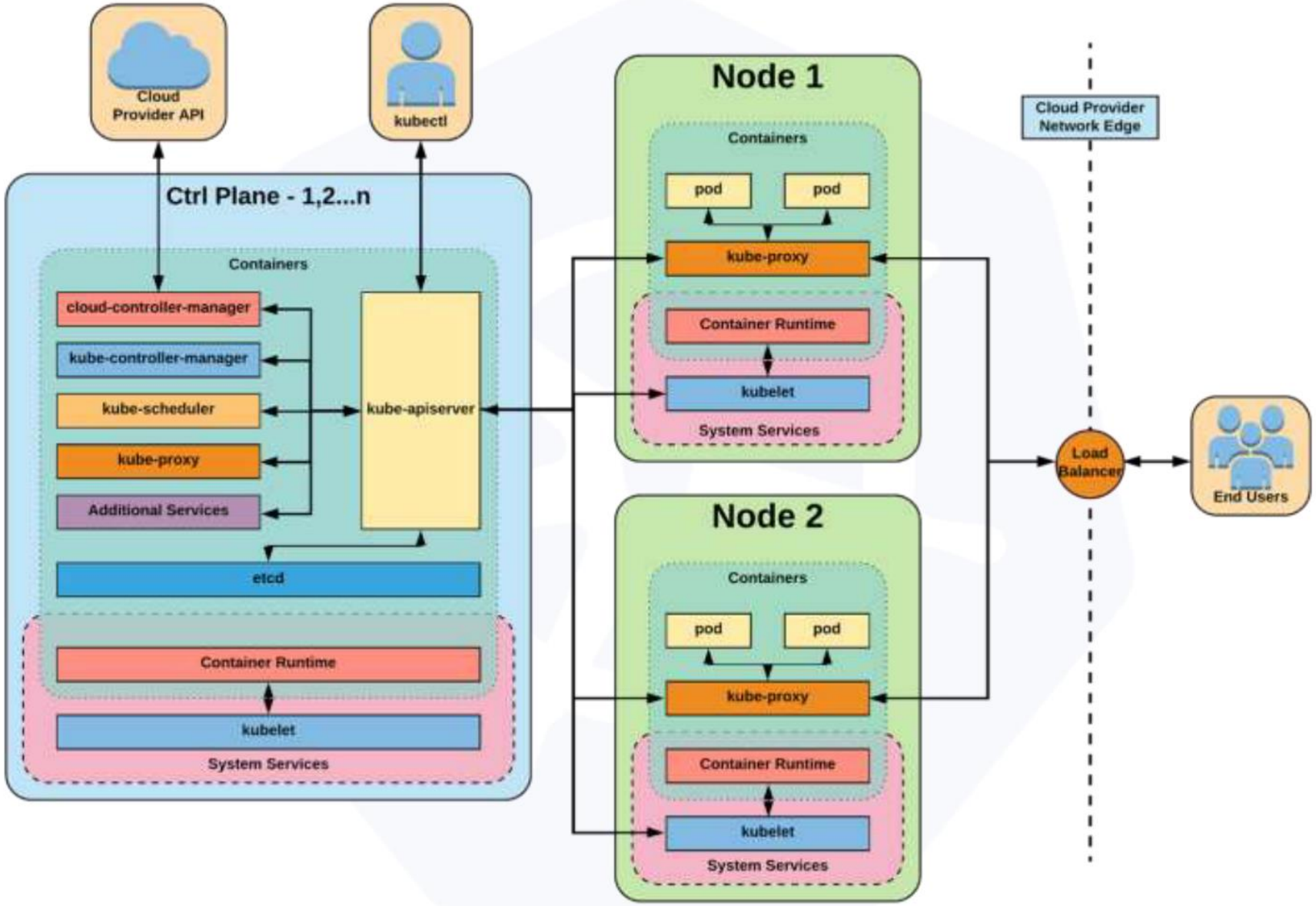


Services

- **Unified method of accessing** the exposed workloads of Pods.
- **Durable resource**
  - static cluster IP
  - static namespaced DNS name

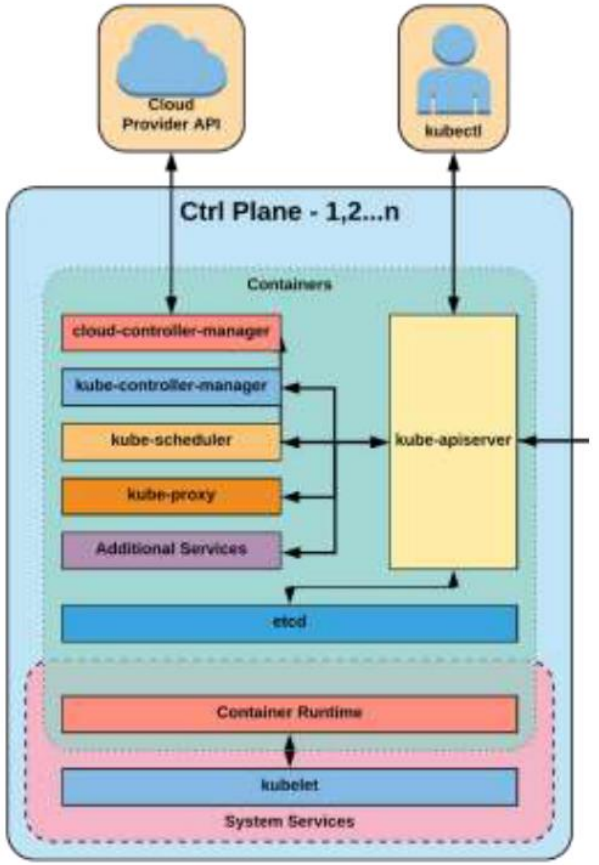


Kubernetes Architecture

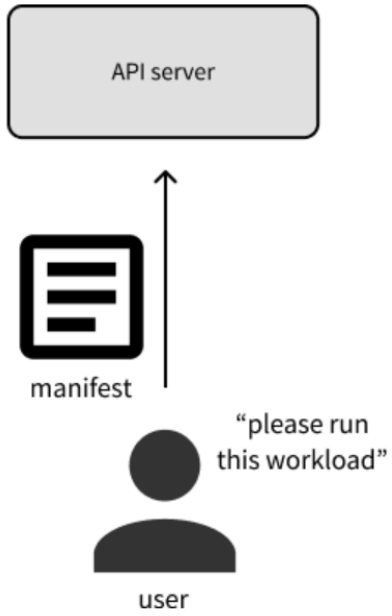


Control Plane

- kube-apiserver
- etcd
- kube-controller-manager
- kube-scheduler
- cloud-controller-manager



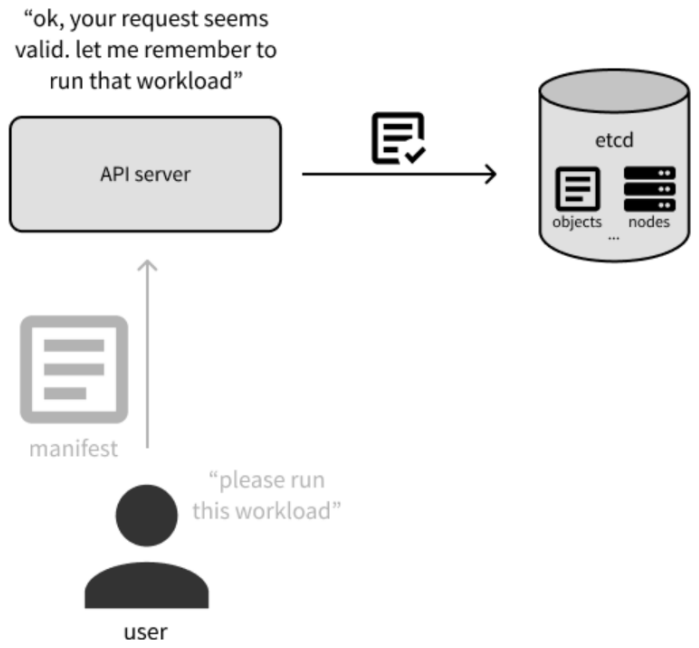
Kube-apiserver



- Provides a forward facing REST interface into the kubernetes control plane and datastore.
- All clients and other applications interact with kubernetes **strictly** through the API Server.
- Acts as the gatekeeper to the cluster by handling authentication and authorization, request validation, mutation, and admission control in addition to being the front-end to the backing datastore.

etcd

- etcd acts as the cluster datastore.
- Purpose in relation to Kubernetes is to provide a strong, consistent and highly available key-value store for persisting cluster state.
- Stores objects and config information.

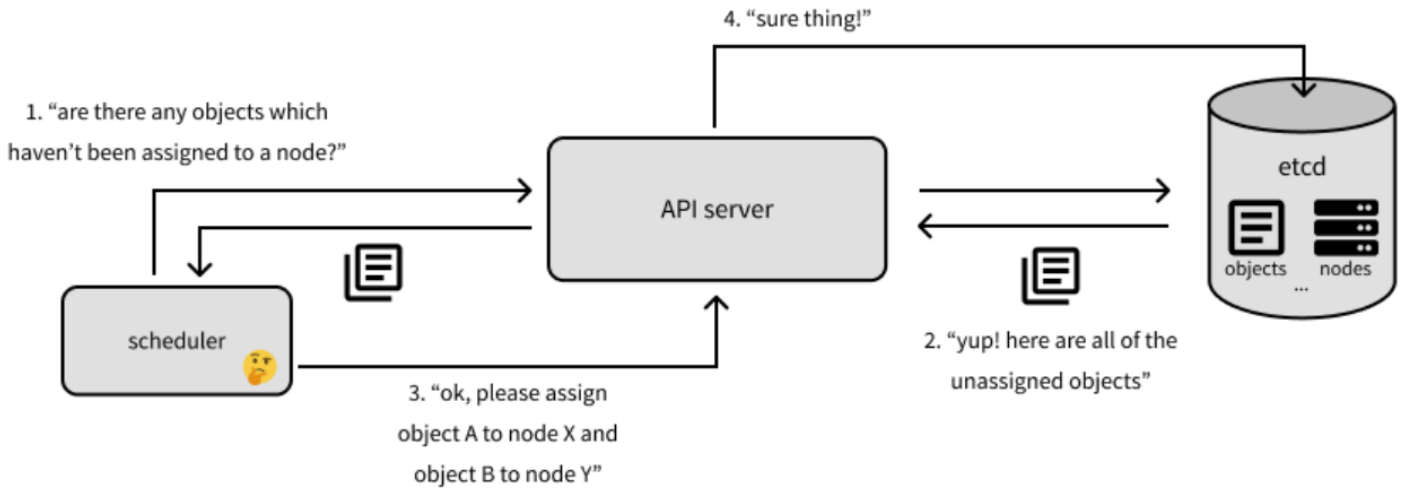


Kube-controller-manager

- Monitors the cluster state via the apiserver and **steers the cluster towards the desired state**.
- Node Controller: Responsible for noticing and responding when nodes go down.
- Replication Controller: Responsible for maintaining the correct number of pods for every replication controller object in the system.
- Endpoints Controller: Populates the Endpoints object (that is, joins Services & Pods).
- Service Account & Token Controllers: Create default accounts and API access tokens for new namespaces.

Kube-scheduler

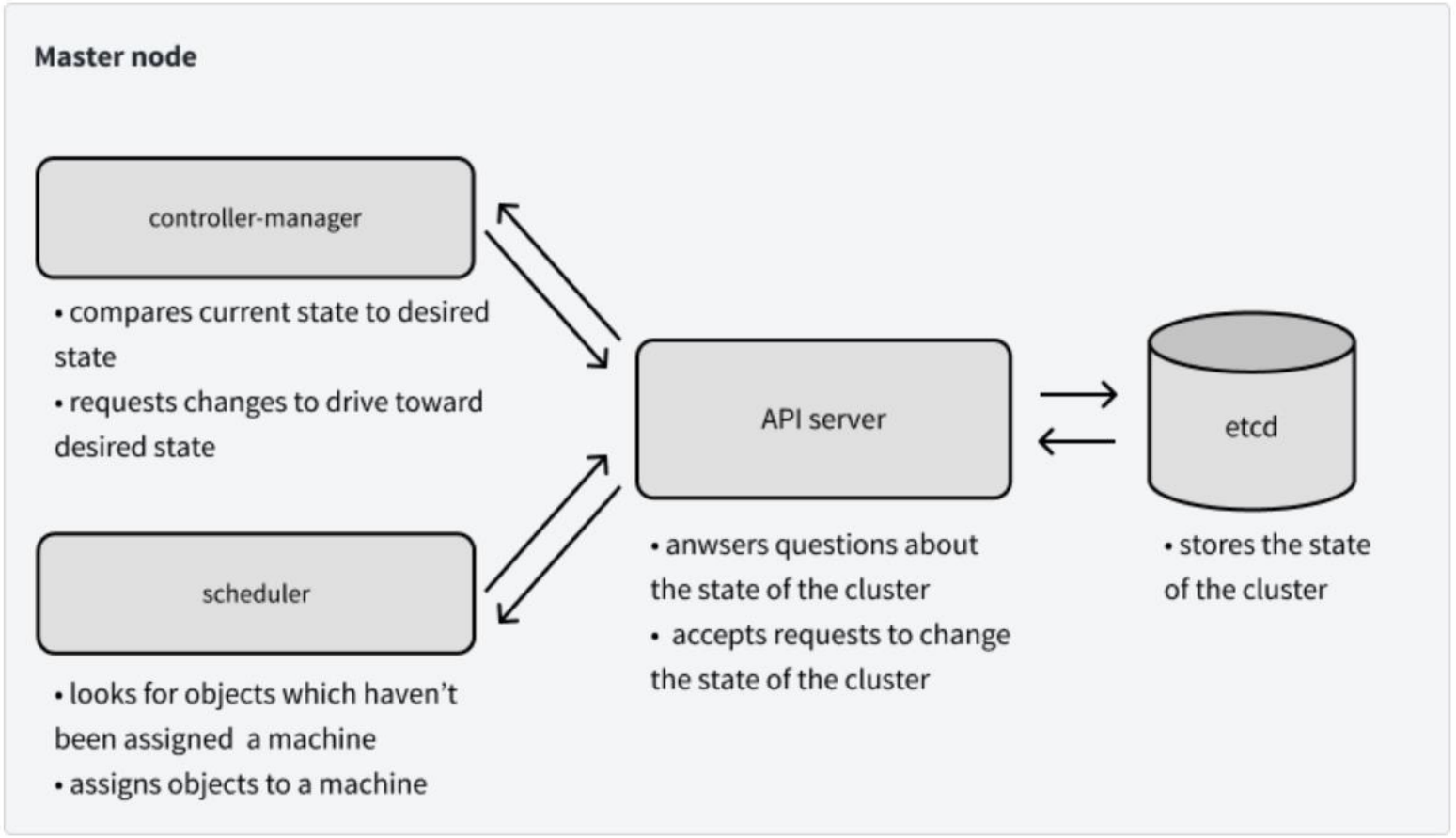
- Component on the master that watches newly created pods that have no node assigned, and selects a node for them to run on.
- Factors taken into account for scheduling decisions include individual and collective resource requirements, hardware/software/policy constraints, affinity and anti-affinity specifications, data locality, inter-workload interference and deadlines.



Cloud-controller-manager

- Node Controller: For checking the cloud provider to determine if a node has been deleted in the cloud after it stops responding
- Route Controller: For setting up routes in the underlying cloud infrastructure
- Service Controller: For creating, updating and deleting cloud provider load balancers
- Volume Controller: For creating, attaching, and mounting volumes, and interacting with the cloud provider to orchestrate volumes

Container Orchestration



Some Terminology

1) A container is launched by running an image. An image is an executable package that includes everything needed to run an application–the code, runtime, libraries, envi-ronment variables, and configuration files. In Docker, this image is called as DockerFile. A DockerFile is always built upon another base image.
   The Docker Hub is a public repository of various base docker images. We can also create our own pub-lic/private repositories on the Docker daemon.
2) Accessing resources like networking interfaces and disk drives is virtualized inside this environment, which is isolated from the rest of the system, post mapped to the outside world, and information about files that must be copied to that environment is provided in the DockerFile.
3) DockerFiles or images are used to "build" containers using the "docker build" command. A container is a runtime instance of an image–what the image is loaded in memory when executed.
4) The build of the application defined in the DockerFile behaves exactly the same wherever it runs. The Docker client communicates with the Docker daemon to create containers. All the images are stored either in the public or private registries. This is shown in the diagram below:

1) Node: It is a worker machine.
2) Cluster: A group of nodes.
4) Pods: They are the atomic units which can contain one or more containers.
5) Deployments: They provide the most common way to create and manage pods.
6) Service: A service declares how pods can be accessed. It is the virtual server inside the node. When pods need to communicate, they use 'services'.

More Terminology

1) *Hosted*

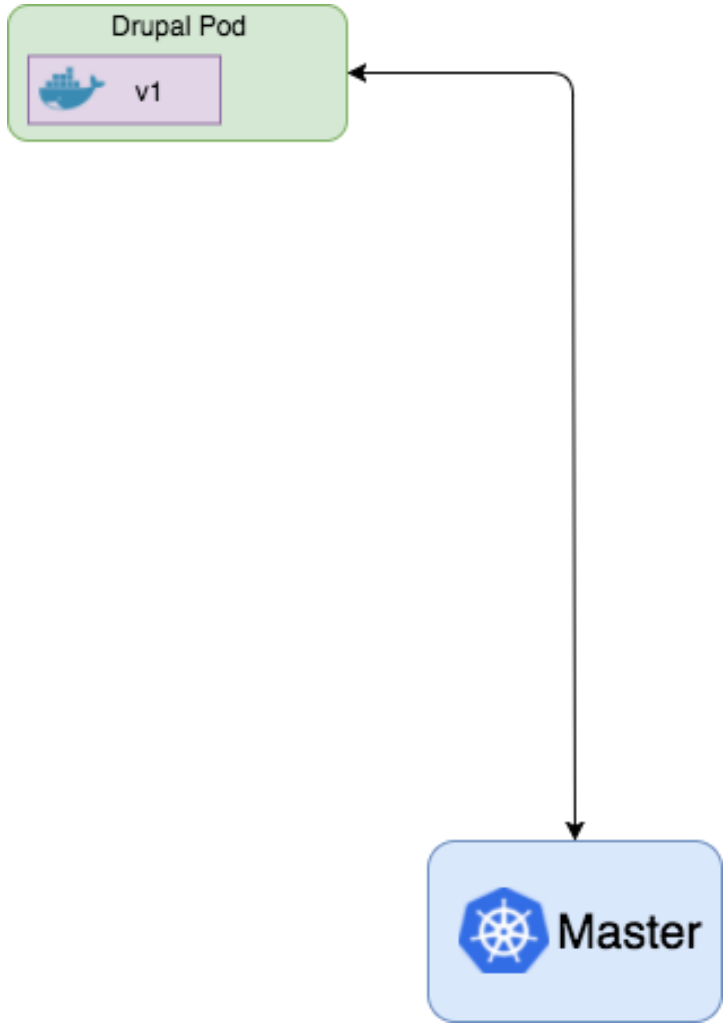
Hosted Kubernetes refers to the usage of cloud ser-vices for working with Kubernetes. It only requires a subscription with a cloud provider (with Kubernetes services) to run and manage containerized applications. It is mainly used by enterprises or DevOps companies during production stages. It is the easier option to choose for scaling machines and to provide greater availability. Some popular hosted Kubernetes services are: Google's cloud Kubernetes Engine, Azure Kuber-netes Service, Amazon's Elastic Container Service for Kubernetes, IBM's Cloud Container Service, etc.

2) *Self-Hosted*

Self-hosted Kubernetes allows a user to run Kubernetes on a single server in order to test/deploy applications with it. It does this by automating steps used to deploy Kubernetes clusters. Kubeadm is the tool used in various distributions of Kubernetes to run locally on a single machine. It is basically used to test/deploy containerized applications with Kubernetes locally before putting it up for production. In Windows, the most popularly used Kubernetes software is MiniKube. MiniKube is essentially Kubernetes running very minimally on a local VM.

Lab 2

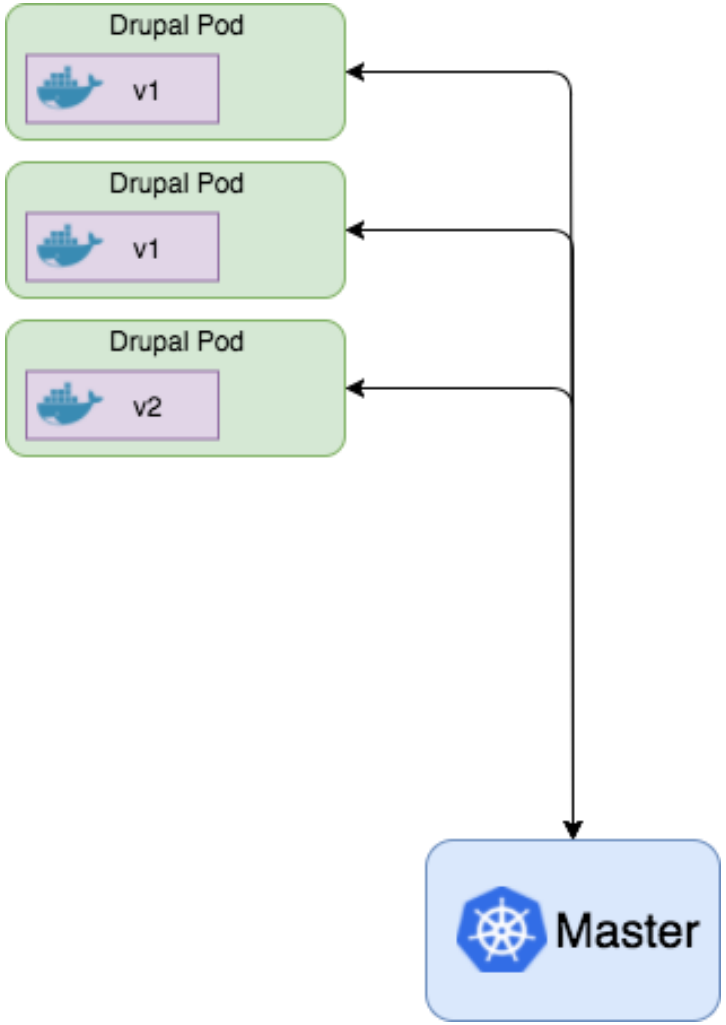
The Pod is the Core Kubernetes Component



- The Pod is the core component of Kubernetes
- Collection of 1 or more containers
- Each pod should focus on one container, however sidecar containers can be added to enhance features of the core container

```
spec:
  template:
    spec:
      containers:
        - name: drupal
          image: cr.io/repo/mydrupal:v1
```

Pods can Handle Scaling and Deployments



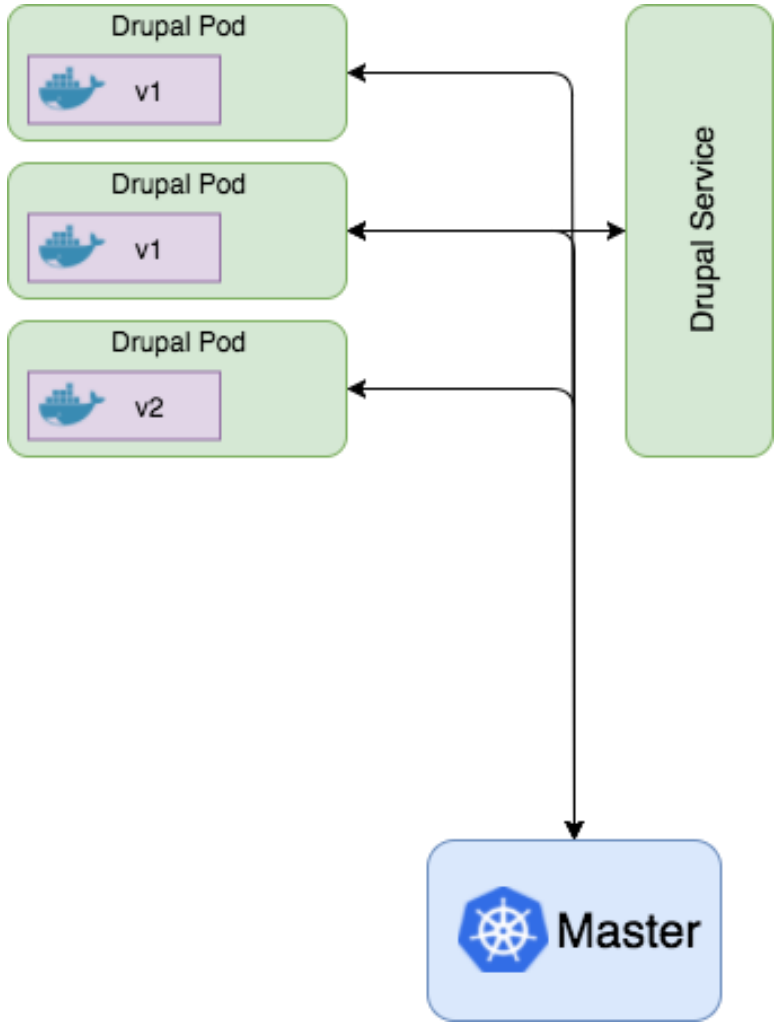
- Once Kubernetes understands what is in a pod, multiple management features are available:

- System Performance
  - Scale up/down the number of pods based on CPU load or other criteria

- System Monitoring
  - Probes to check the health of each pod
  - Any unhealthy ones get killed and new pod is put into service

- Deployments
  - Deploy new versions of the container
  - Control traffic to the new pods to test the new version
    - Blue/Green deployments
    - Rolling deployments

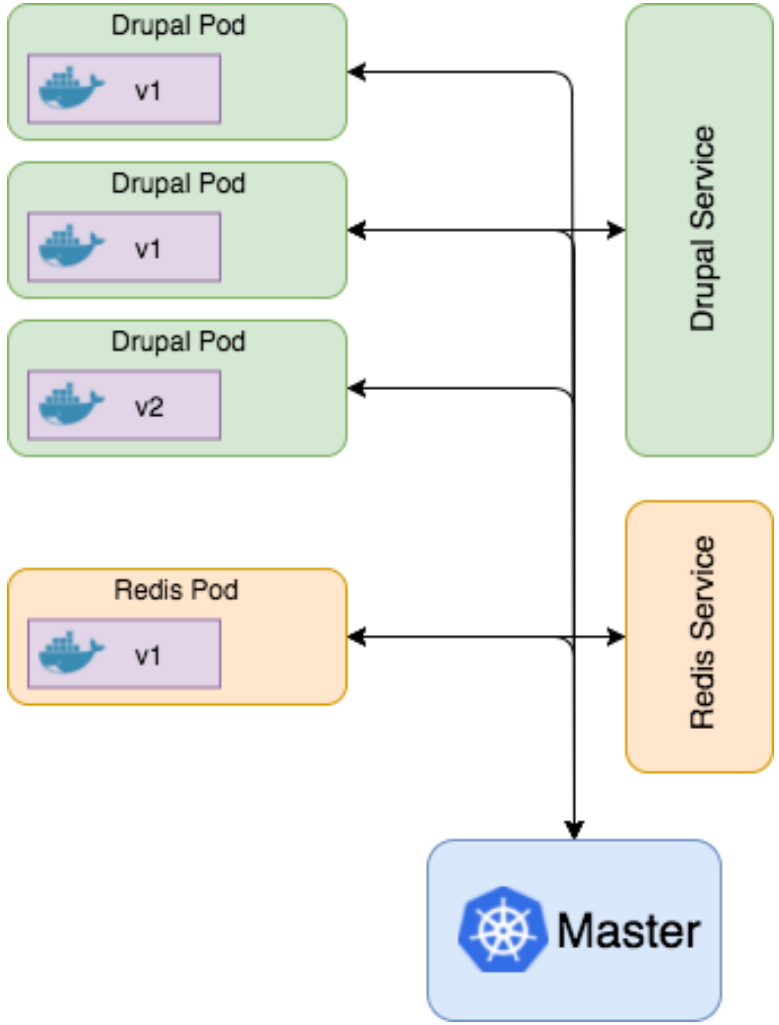
Kubernetes Services tie together the pods



- Kubernetes Services are used to control communications with the pods
  - Load balance the requests
  - Don’t send traffic to the unhealthy ones
  - Only talk to the correct version

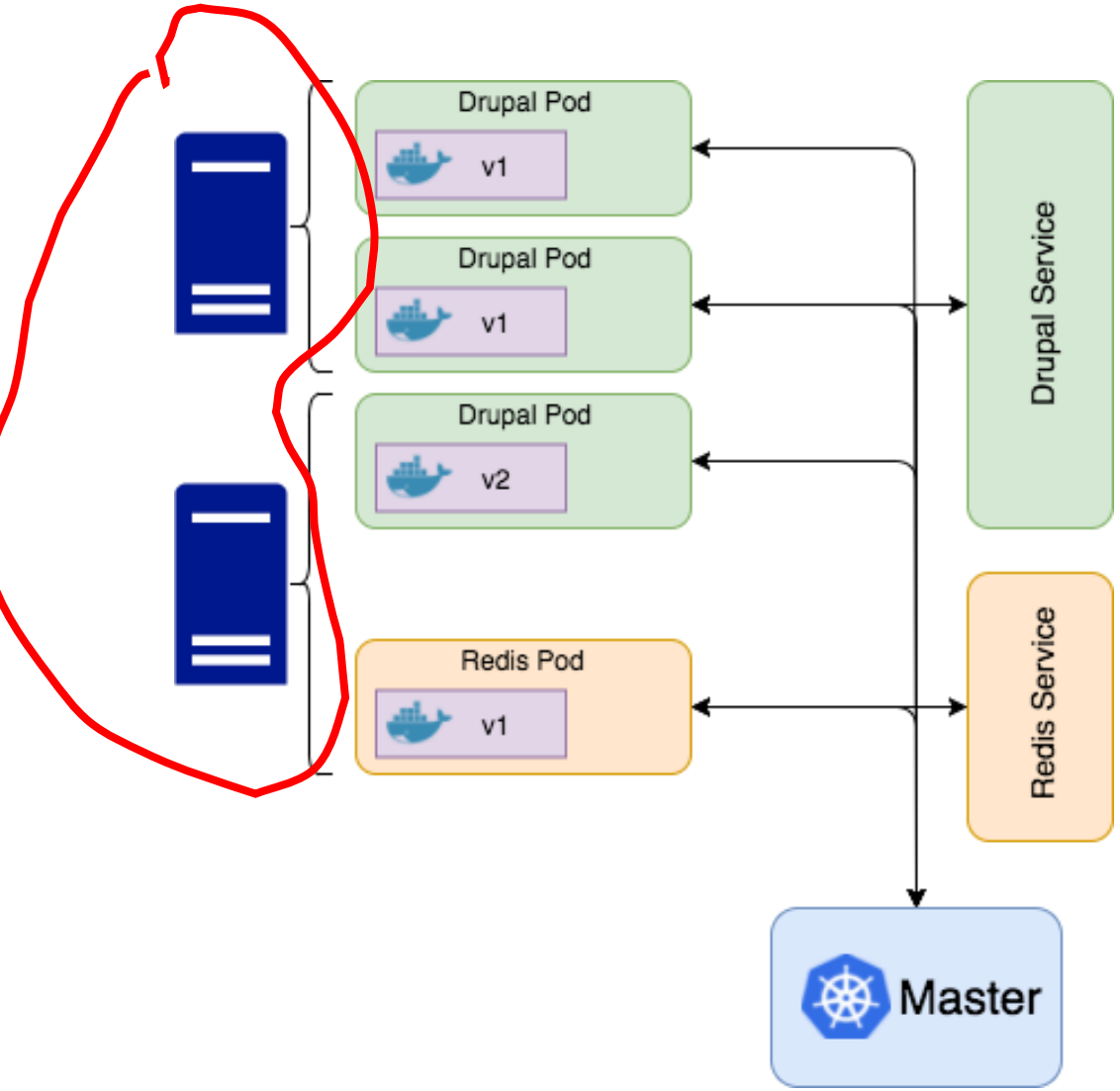
```yaml
apiVersion: v1
kind: Service
metadata:
  name: drupal
spec:
  selector:
    app: drupal
  ports:
    - name: http-port
      port: 80
  type: LoadBalancer
```

Services Structure allow multiple Components



- With the Service architecture Kubernetes handles things that you often might have to worry about
  - Service discovery
  - Load balancing
  - Scaling
- Service discovery allows each pod just needs to call the name of the service it wants to talk to
- Services have multiple options
  - Session based load balancing
  - Single port based services
  - External Services
- The Service architecture of Kubernetes can be scaled up to handle as many services as you would like for your system

Where Is the Infrastructure?



- You don’t have to worry about the infrastructure
- The entire design of pods and services is described with YAML files
- Nothing in deployments, pod management, service discovery, monitoring, etc required any knowledge about how many servers, IP addresses, load balancers, or anything else with the infrastructure
- Behind the scenes, Kubernetes is aware of all of the servers available, load balancers, application gateways and will configure them automatically according to what is in the YAML files

Deployment

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: drupal
spec:
  template:
    spec:
      containers:
        - name: drupal
          image:
cr.io/repo/mydrupal:v1
          ports:
            containerPort: 80
```

- Deployment—connects a Pod with replication control and rollout management
  - Synchronizes app configuration across instances
  - Production deploys are as simple as updating an image tag
  - No more bouncing apache on a dozen servers
- Contains a Pod spec

Autoscaling

```
apiVersion: autoscaling/v1
kind: HorizontalPodAutoscaler
spec:
  scaleTargetRef:
    apiVersion:
extensions/v1beta1
    kind: Deployment
    name: drupal
  minReplicas: 2
  maxReplicas: 10
  targetCPUUtilizationPercentag
e: 50
```

- Realizes the promise of the cloud: scales your app in response to load, in real time
- Kubernetes tracks resource utilization
- Responds by adding or removing pods to the Replica Set
- Kubernetes core supports CPU utilization
- Other resources are available via add-ons

Service

```yaml
apiVersion: v1
kind: Service
metadata:
  name: drupal
spec:
  selector:
    app: drupal
  ports:
    - name: http-port
      port: 80
  type: LoadBalancer
```

- `curl http://drupal/cron.php`
- Manages ports and internal IP's with domain name resolution
- Opens ports on agent nodes
- Manages load balancing between pods
- Provisions cloud provider load balancer
- Exposes pods to Kubernetes service discovery

External Service

```
kind: Service
apiVersion: v1
metadata:
  name: mysql-service
spec:
  type: ExternalName
  externalName:
mysql.example.com
  ports:
    - port: 3306
```

- Use RDS and provider services when possible
- No need to hard code external services in your application
- Adds an external resource to Kubernetes service discovery

Deployment: Configuration Management

```yaml
apiVersion: extensions/v1beta1
kind: Deployment
spec:
  replicas: 2
  template:
    spec:
      containers:
        - name: drupal
          image: cr.io/repo/mydrupal:v1
          ports:
            containerPort: 80
          env:
            - name: DB_HOSTNAME
              value: mysql-service
            - name: DB_PASSWORD
              valueFrom:
                secretKeyRef:
                  name: mysql-service-secrets
                  key: password
      imagePullSecrets:
        - name: registrykey
```

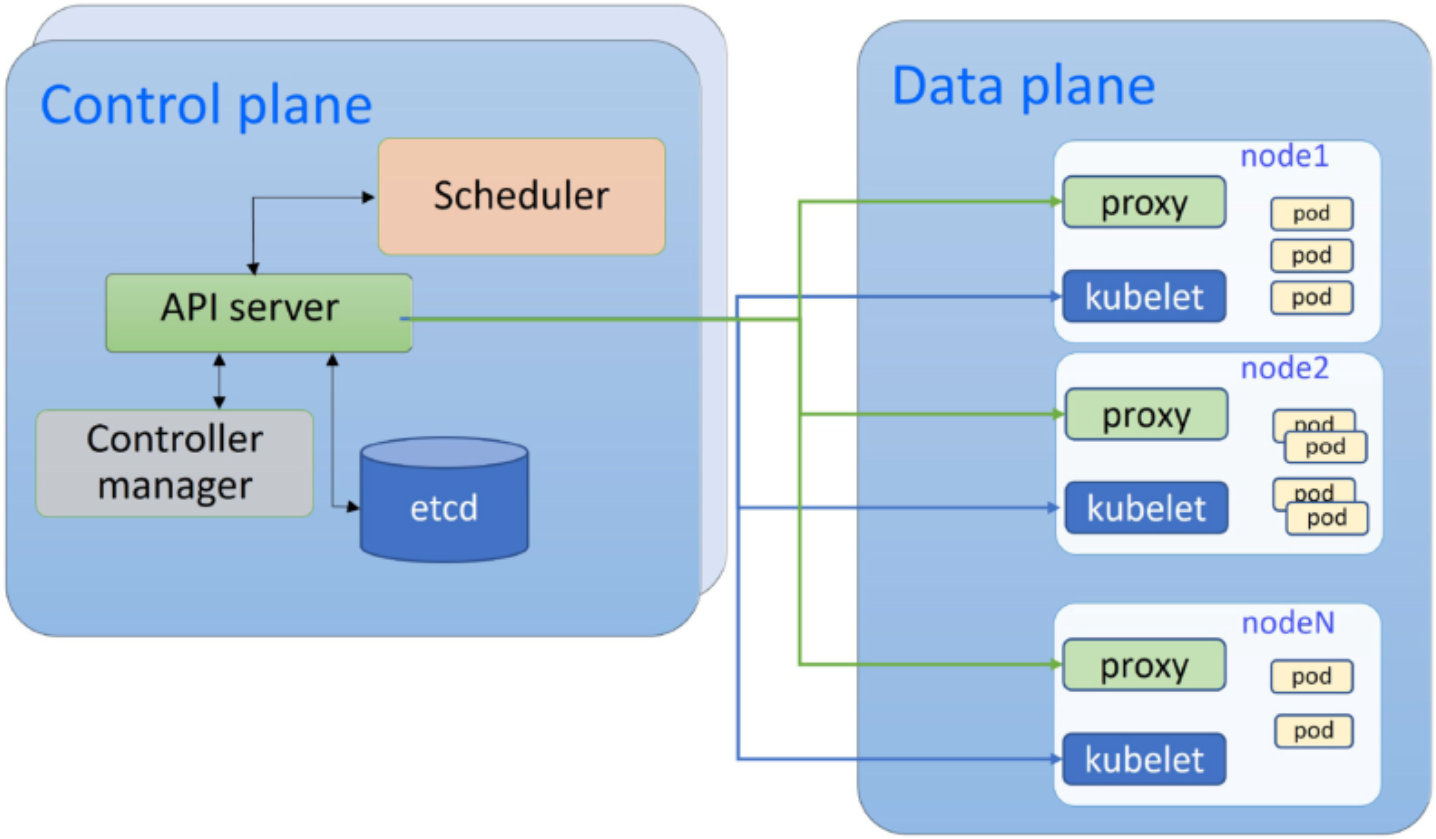
```
 *   $databases['default']['default'] = array(
 *     'driver' => 'sqlite',
 *     'database' => '/path/to/databasefilename',
 *   );
 * @endcode
 */
$databases['default']['default'] = array(
  'driver' => 'mysql',
  'database' => 'mydrupaldb',
  'username' => getenv('DB_USERNAME'),
  'password' => getenv('DB_PASSWORD'),
  'host' => getenv('DB_HOSTNAME'),
);

/**
 * Access control for update.php script.
 *
 * If you are updating your Drupal installation using
 * are not logged in using either an account with the
 * updates" permission or the site maintenance
account
 * created during installation), you will need to
modify
```

Deployment: Volumes

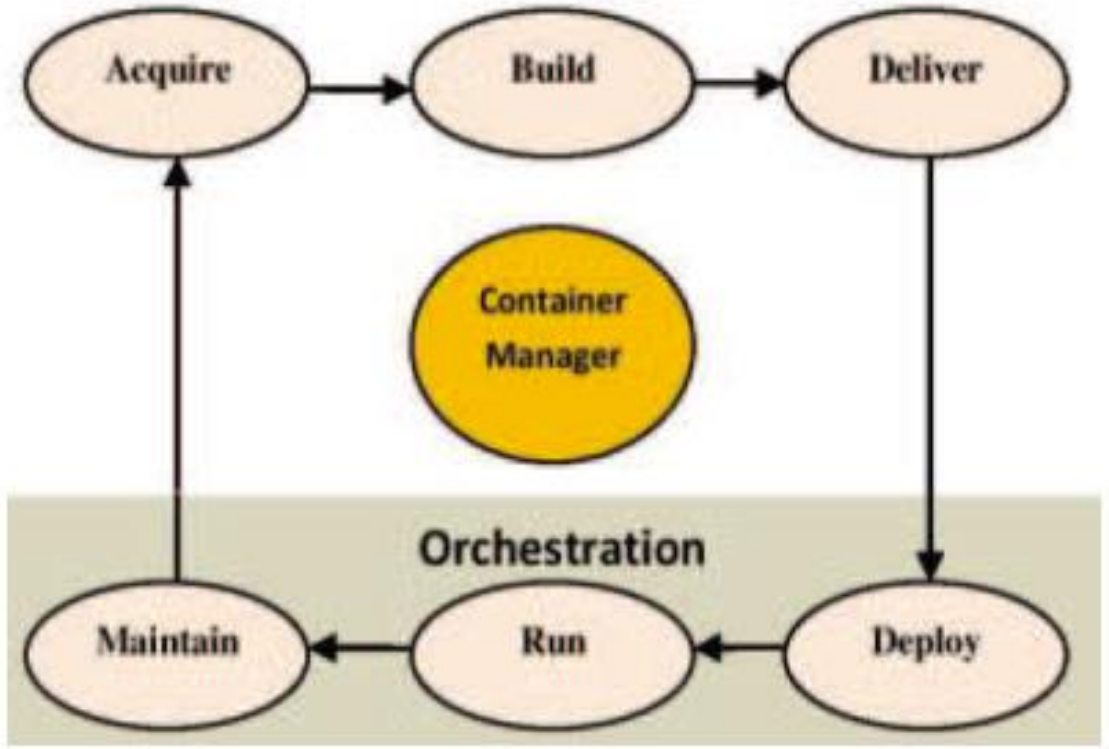
```
apiVersion: extensions/v1beta1
kind: Deployment
spec:
  replicas: 2
  template:
    spec:
      containers:
        - name: drupal
          image: cr.io/repo/mydrupal:v1
          ports:
            containerPort: 80
          volumeMounts:
            - name: my-drupal-volume
              mountPath: /drupal-
7.56/sites/files
      volumes:
        - name: my-drupal-volume
          azureFile:
            secretName: azure-storage-secret
            shareName: <pre-existing-file-
share>
            readOnly: false
```

- Manages networked drives across containers and VM's
- `volumeMounts` sets the mount path and references a named volume
- Volumes can be defined as
  - Pre-created named volumes
  - Dynamically provisioned Persistent Volume Claims

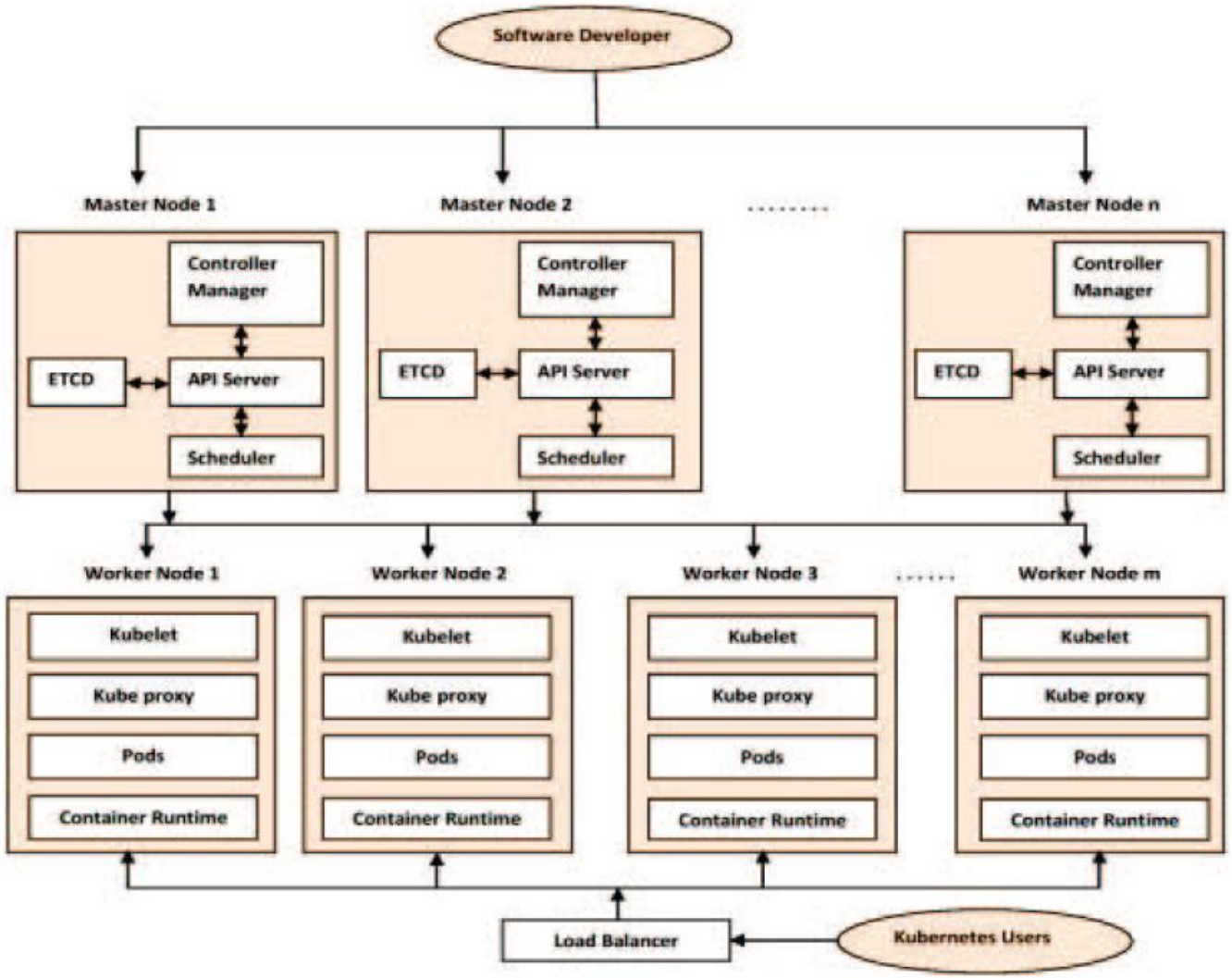
K8S architecture



Life Cycle of Container



Full Fledged K8S Deployment



Different Features of Orchestration

Feature	Technique	Function
Scheduling	kube-scheduler, custom	Assign containers to worker nodes based on different policies and user specifications.
Admission control	mutating and validating controllers, custom	Intercept requests to the K8s API server before the object is persisted, but after the request is authenticated and authorized. This increases the security of the cluster.
Load balancing	round-robin, custom	Distribute the load among multiple container instances. Complex policies can be provided by external load balancing.
Health check	start-up, liveness, readiness, and shutdown probes	Control whether a container can answer to requests.
Fault-tolerance	replica controller, custom	Specify and maintain a desired number of containers.
Auto-scaling	horizontal POD autoscaler, custom	Reshape the K8s workload by automatically increasing or decreasing the number of pods. Custom metrics can be used.

Scheduler IaaC

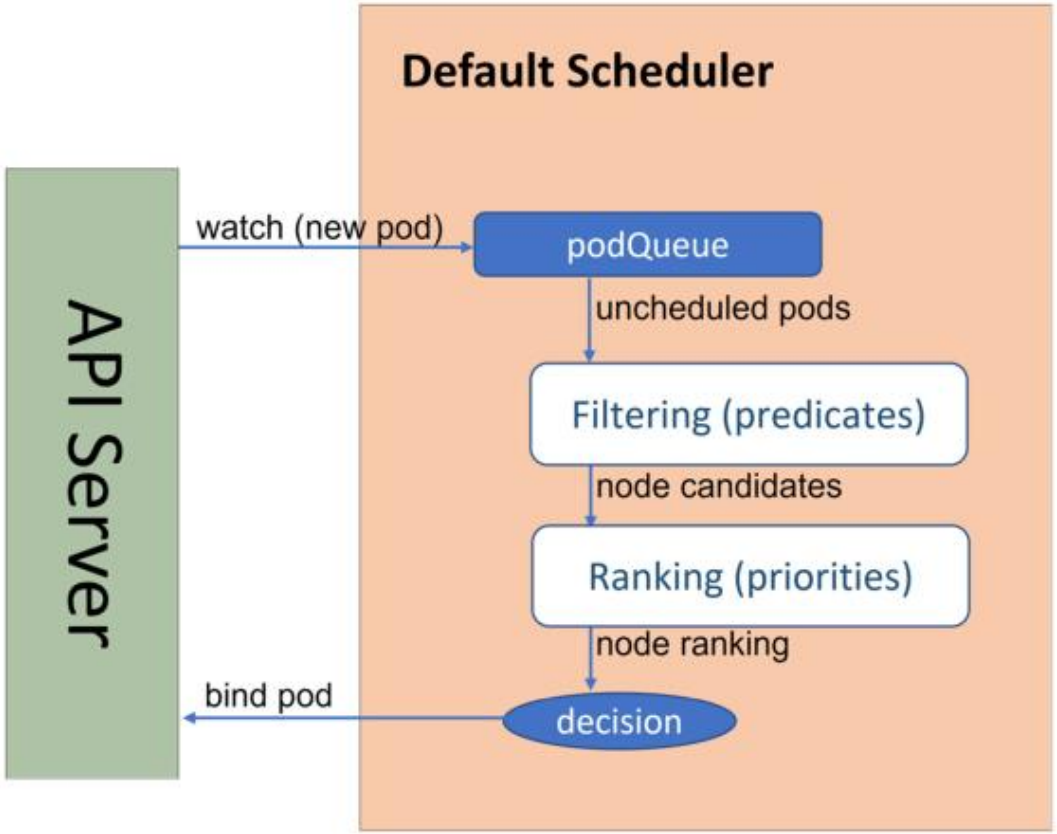
```
apiVersion:kubescheduler.config.k8s.io/v1beta2
kind: KubeSchedulerConfiguration
profiles:
  - schedulerName: default-scheduler
  - schedulerName: new-scheduler
    plugins:
      score:
        disabled:
        - name: NodeResourcesLeastAllocated
        enabled:
        - name: NodeResourcesMostAllocated
          weight: 3
```

Listing 1. Defining profiles for kube-scheduler.

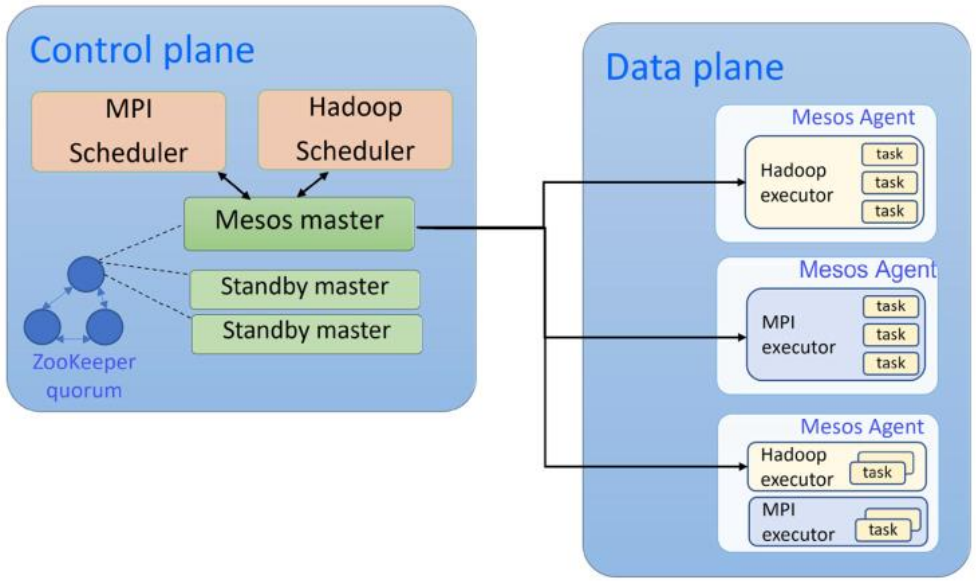
```
---
apiVersion: v1
kind: Pod
metadata:
  name: mynginx
spec:
  schedulerName: new-scheduler
  containers:
  - name: mynginx
    image: nginx:1.15.11
    ports:
    - containerPort: 80
```

Listing 2. Assigning a scheduler to a pod.

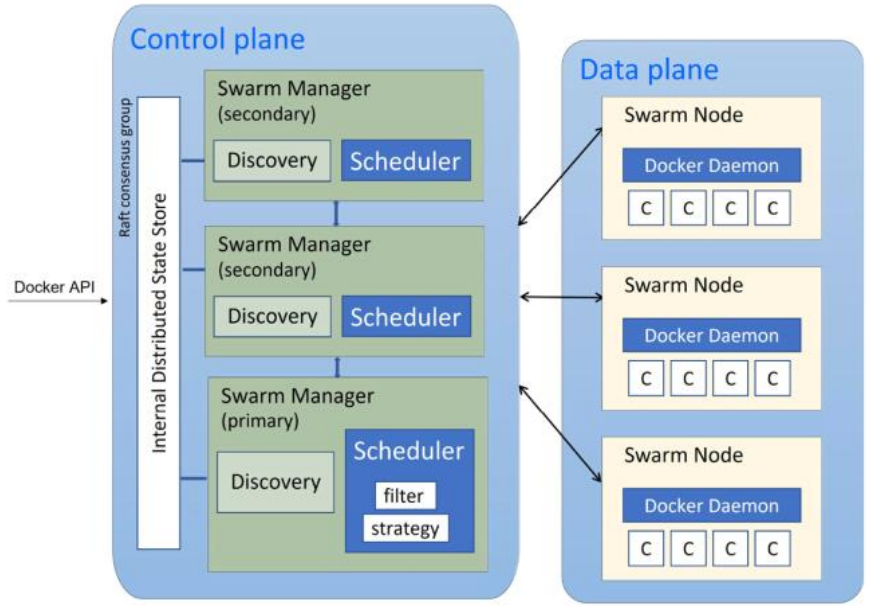
How the Scheduler Works



Other Common Schedulers (2-Level Schedulers)



(a) Mesos Architecture.



(b) Docker Swarm Architecture.

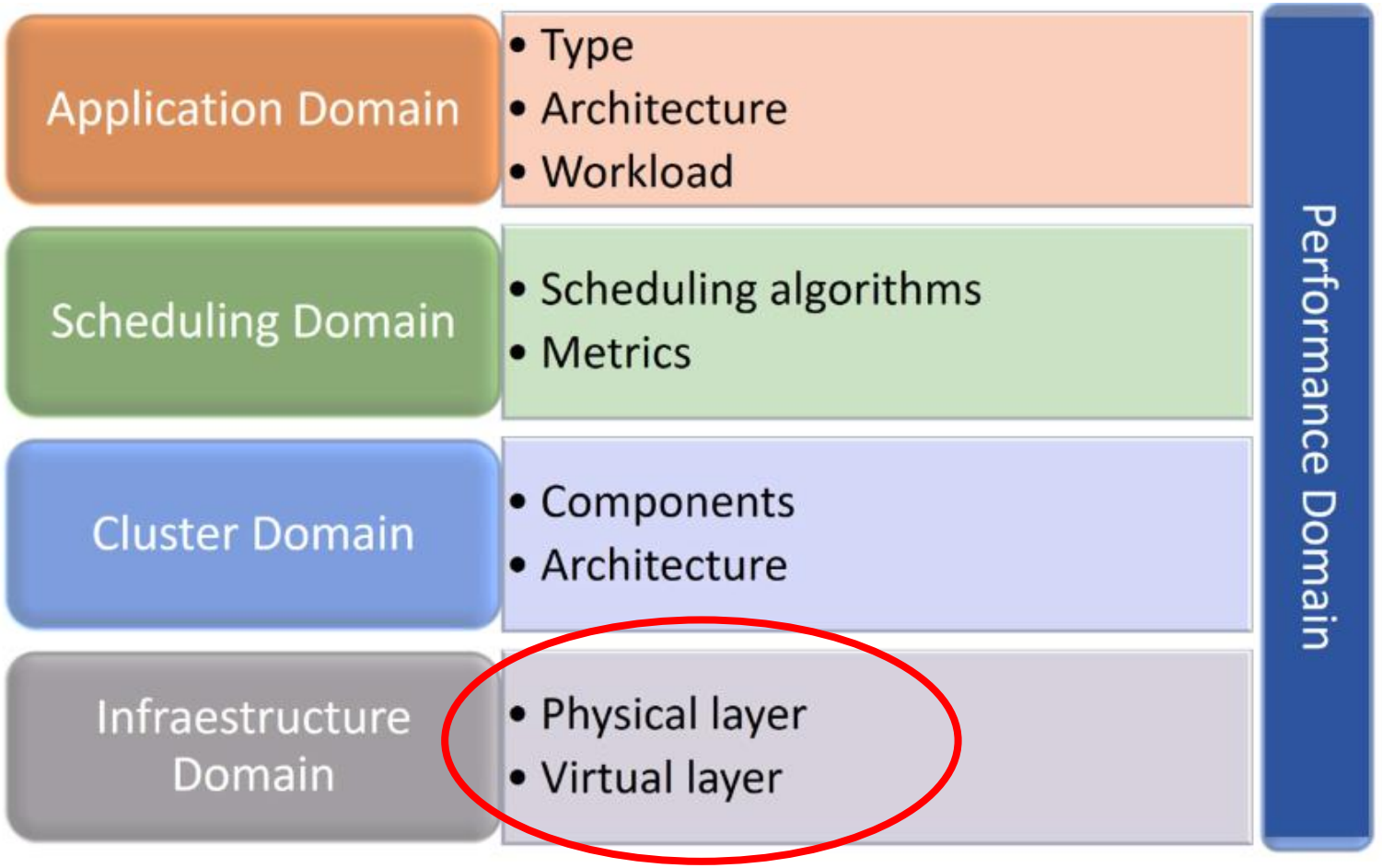
Different Scheduler Options

Framework	Container technology	Scheduler architecture	Scheduler algorithm	Applications
Kubernetes	CRI API, OCI-compliant	cluster centralized	round-robin dispatch, filtering and ranking	all (multiple co-located tasks)
Borg	cgroup-based	monolithic centralized	priorizated round-robin	all (indepen. tasks)
Apache Mesos	mesos contain docker, singularity	two-levels	DRF, framework-based	all (single-task)
Hadoop YARN	cgroups-based, docker	two-levels	FIFO round-robin, fair capacity preemptive	batch apps
Docker Swarm	docker	cluster centralized	filtering and scheduling (spread-binpack-random)	long-running apps

Details of Different K8S Schedulers

S. No.	Parameters	Kubernetes	Docker Swarm	Mesos	OpenShift
1.	Initial Release Date	July 2015, V1.16 in Sept. 2019	March 2013, Stable release July 2019	July 2016, Stable release August 2019	4th May 2011, Stable release Oct. 18, 2021
2.	Deployment	Almost any platform	Linux, Windows, and macOS	Ubuntu, Debian Jessie, CentOS	RHELAH, Fedora, or CentOS
3.	Security	Does not come with built-in authentication or authorization capabilities	Based on Network Level Security	Does not provide authentication by default	Strict Security policies
4.	Maturity/Stability	Very mature	Mature but still evolving	Very mature, especially for very clusters counting in the thousands of servers	Evolving
5.	Scalability	Medium to large clusters	Small to medium scale clusters	Large to Very large scale clusters	Small to medium scale clusters
6.	Cluster Installation	Slightly complex to set up.	Very easy to install and setup	Generally easy to install and set up but considerably more complex with large setups.	Easy to setup
7.	Best features	Best pods scheduling features	Easy to use and more native to docker	Scale in 1000s and Rack/Host-based constraints	Easy scaling and deployment
8.	Images Supported	Docker and rkt, restricted	Docker-image format	Mostly Docker-image	OCI, Docker-container image
9.	Learning Curve	Rapid	Easy	Rapid	Easy
10.	Support	Large active community	-	-	Small community
11.	Product vs. Project	Open-source project	Open Source Plateform	Open Source Framework	Open Source, Commercial Product

Extensive Work Done on K8S Orchestration



Physical Infrastructure Layer

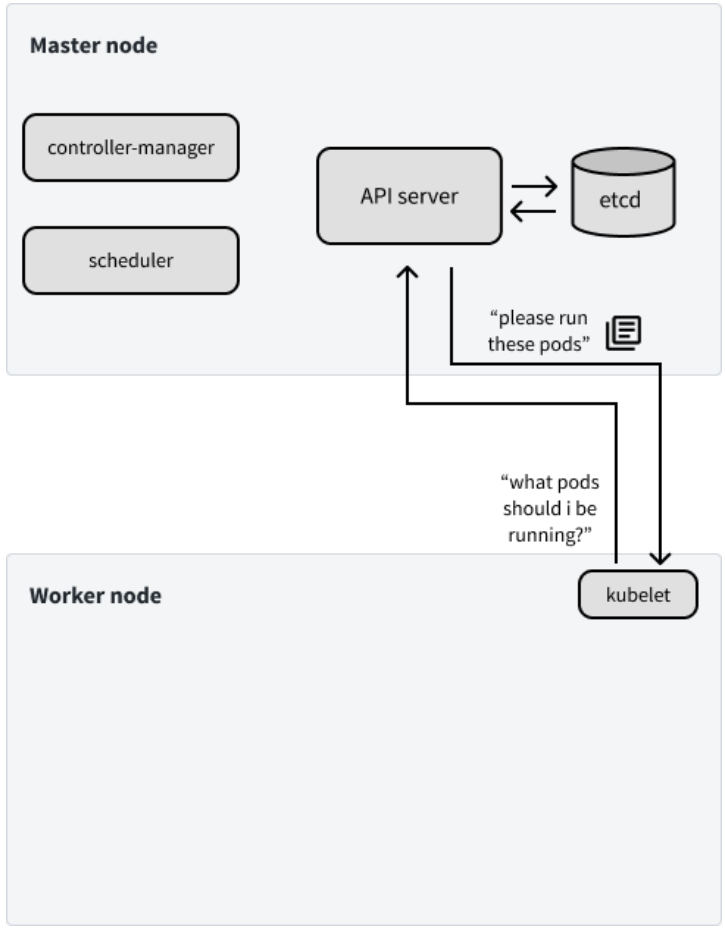
Fe.	Proposal
hardware	The authors tackle heterogeneous nodes with different CPU architectures, memory, and network connectivity by extending K8s with a monitoring tool. The proposed dynamic scheduler can reschedule stateless applications and provide failure recovery in case of a Raspberry Pi node failure.
	The authors propose to take the physical, operational, and network parameters into consideration, along with the software states to orchestrate edge-native applications dynamically on 5G. The approach extends the limits of the nodes' capabilities.
	The authors present an intelligent K8s scheduler that takes into account the virtual and physical infrastructure of complex data centers. More precisely, a server hardware behavior model is generated using a wind tunnel. Results carried out in real data centers show that introducing hardware modeling into the scheduler maximizes effectiveness (around 10–20%).
	Tarema is a system that addresses task and infrastructure profiling to dynamically assign scientific workflow tasks to heterogeneous cluster resources. Groups of similar worker nodes are created considering the hardware properties of the nodes. Then, K8s uses node groups and task labels to allocate tasks to available cluster resources at runtime. Tarema reduces the runtimes of real-world workflows compared to the popular shortest job next and shortest jobs to the fastest resources schedulers (around 4.54%) while providing a fair cluster usage.
	Polaris is built on top of KubeEdge and is sensitive to the computational capacity, hardware capacity, and energy efficiency of edge nodes. The Polaris scheduler also takes into account device locality and mobility by abstracting a cluster topology.
GPAs	Gaia scheduler allocates GPU resources for Nvidia brand and a K8s plugging is used to divide a physical GPU into multiple GPUs and assign these virtual GPUs to a container.
	The authors built a GPU-aware resource orchestration layer, called Knot, and integrated it with K8s. Kube-Knot proposes a scheduler that leverages real-time GPU usage correlation metrics to perform safe pod co-locations, ensuring crash-free dynamic container resizing on GPUs.
	The authors propose an adaptive GPU scheduler that allocates and reallocates GPUs based on a sidecar that monitors the training process on the GPU.
	The authors present a machine learning-based scheduler to select a CPU or GPU hardware device in a K8s environment. The algorithm creates a model of runtime prediction of the applications and is also based on the nature of the processed data.
	The authors extend the K8s scheduler to manage the usage of hardware-accelerated nodes in **multi-access edge computing** (**MEC**) architectures. The scheduler uses information about the previous executions of a particular application to select the best CPU or GPU node.

Virtual Layer

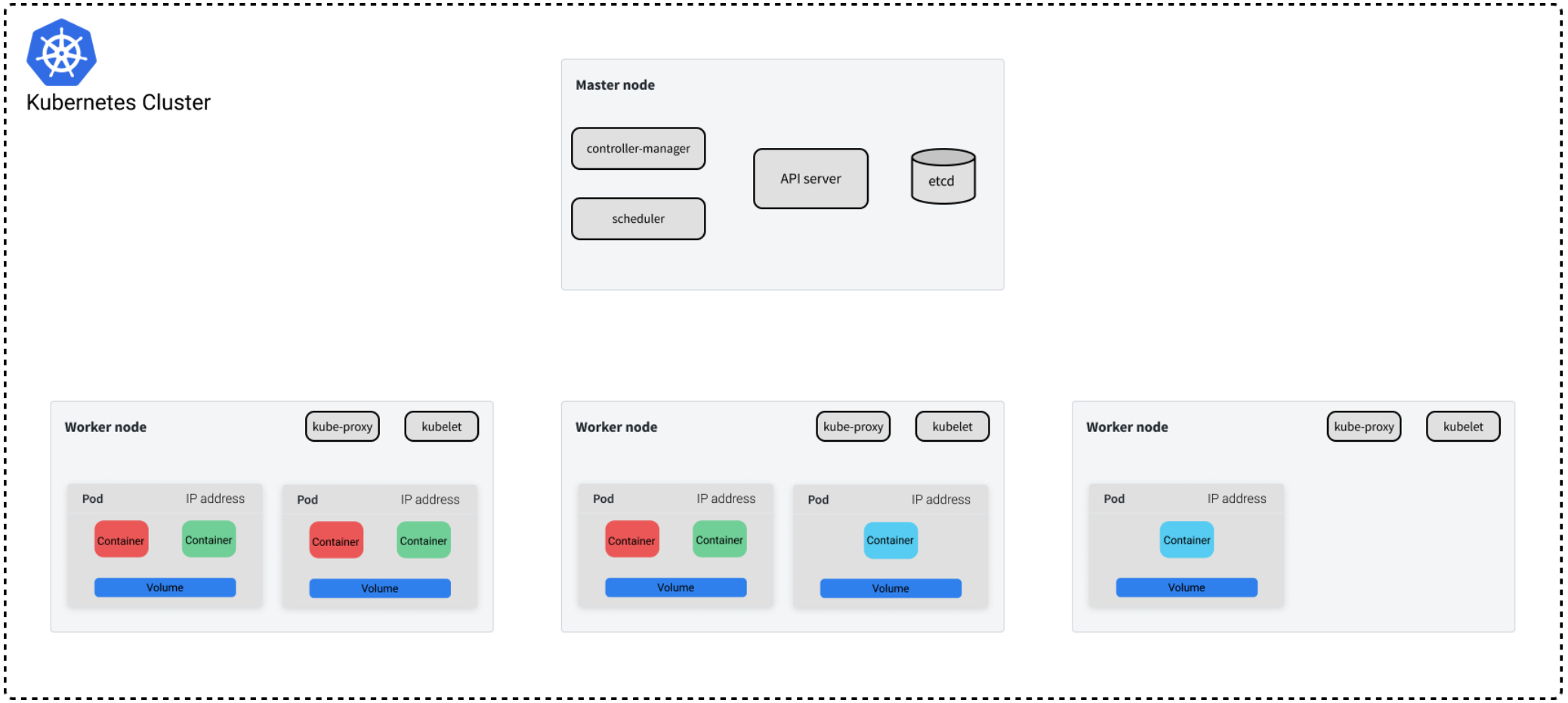
Fe.	Proposal
interference	The authors develop a reference net-based model (a kind of Petri net) for resource management within K8s, to better characterize performance issues and mitigate the effects of interference.
	KubeSphere presents a multi-tenant policy-driven meta-scheduler to provide fairness among competing users.
	The authors design an interference minimization model coupled with minimum energy utilization and maximal green energy consumption for scheduling industrial IoT applications. The proposed scheduler leads to improved energy utilization and minimal interference around 14% and 31%, respectively, in contrast to a standard first-come first-served scheduler.
	The authors propose a scheduling algorithm oriented to a multi-tenant model, in which team users are modeled as virtual clusters and cluster load is monitored regularly. The optimized K8s scheduling algorithm is applied to Docker-based deep learning platforms to ensure load balancing.
	The authors address the problem of interference in the multi-tenant cloud and design an interference-aware custom scheduler to be deployed on top of a virtualized environment. The technique extracts the system, socket, and core metrics to model the hardware (i.e, IPC, reads, writes) and the applications (i.e., reads, writes). Results showed that the custom approach improves the overall performance of the workloads while achieving a more balanced resource utilization than the default K8s scheduler.
	The authors extend the K8s scheduler to use a QoE-aware container scheduler and a re-scheduler for video streaming applications. They create a machine learning predictor that estimates the subjective **Mean Opinion Scores** (**MOS**) for a video session using input resource usage metrics from the worker nodes and the containers. The rescheduling algorithm is triggered in case of QoE degradation in the container due to co-location with other services.
	The authors evaluate the performance interference between CPU-intensive and network-intensive containers, and between multiple network-intensive containers. Results show that containers experience performance degradation by 50% due to the co-located containers.
	The proposed Skynet resource management approach builds a model at runtime to map a target performance-level objective for each application to resources, taking into account multiple resources and changing input load. Skynet decreases performance-level objective violations (7.4×) and increases resource utilization (2×)compared to K8s.
network	The authors present a network-aware scheduler that deploys a service in the node that minimizes the **Round Trip Time** (**RTT**) as long as it has enough bandwidth for the service.
	ElasticFog collects real-time network traffic information and dynamically allocates resources based on this information. It adapts to changes in the state of network traffic over time to minimize latency and maximize the throughput of client requests.
	Polaris scheduler creates a cluster topology graph to make the scheduling aware of the state of the network infrastructure for IoT service on the edge.
	The authors present a network-aware scheduler that selects the node most suitable for an application within a given deadline considering the RTT and the bandwidth of the network.
NVF	The authors present a scheduler to perform the optimal placement for service instances into a logical network by selecting the appropriate VNFs along with their service graph. The evaluations show that this design is very promising for the 5G slicing with a large-scale orchestration system.
	Megalos is presented as an ETSI NFV compliant architecture that supports the implementation of virtual network scenarios consisting of several networking components (VNFs). The scheduling allocates VNFs to the nodes taking into account the network topology to reduce the traffic among nodes. The proposal guarantees the segregation of each virtual LAN traffic.
	The authors design and implement a **Service Function Chain** (**SFC**) controller as an extension to the K8s scheduling. The SFC controller optimizes the placement of service chains in Fog environments, specifically tailored for Smart City use cases. The approach reduces the network latency when compared to the default scheduling mechanism.
	The authors implement K8s function scheduler plugins to deploy VNFs to nodes according to the results obtained from multiple reliable allocation models. The model is selected depending on user objectives, and new models can be added to the scheduler without restarting the scheduler.

Cluster Domain

Fe.	Proposal
cloud	The paper presents a platform that includes a comprehensive monitoring mechanism and a dynamic resource-provisioning operation
	OASIS is an online algorithm for admitting and scheduling asynchronous training jobs in a machine learning cluster. The number of concurrent workers and the server parameters for each job are dynamically adjusted to maximize the overall utility of all jobs based on their completion times
	The paper describes a multi-level scheduling solution for mobile network slicing
	Pigeon is a FaaS framework for private cloud that uses function-level resource scheduling and oversubscription-based static pre-warmed container pool to increase system performance
	The paper proposes a multi-resource fair scheduling algorithm for independent batch jobs among multiple federated cloud computing
	The paper presents an autonomic, **Adaptive Bin Packing** (**ABP**) algorithm to assign virtual entities to physical devices without the need to set any configuration
	The paper presents a deep reinforcement learning-based job scheduler for scheduling independent batch jobs among multiple federated cloud computing clusters adaptively
	The paper optimizes the calculation of the number of pods for Knative, a platform for serverless workloads, based on Double Exponential Smoothing
	The paper investigates how containers are distributed on top of virtual machines when a complex application refactored as microservices is deployed in a private cloud environment, in which containers are distributed on top of VMs
	The paper proposes a scheduler extension and resource rescheduling that incorporates the **quality of experience** (**QoE**) metric proposed in the ITU P.1203 standard into **Service Level Objectives** (**SLOs**). A predictor based on machine learning estimates the QoE offered by the cloud
	The paper presents a self-adaptive K8s cloud controller for scheduling time-sensitive applications
fog	Foggy is an architectural framework and a software platform for workload orchestration and resource negotiation in a multi-tier and distributed system
	DYSCO can run and reschedule stateless applications providing failure recovery in case of a node failure
	The paper proposes a custom scheduler for K8s orchestrator that distributes the scheduling task among the processing nodes by means of a MAS
	The paper proposes a network-aware scheduling approach for container-based applications in Smart City
	ElasticFog runs on top of K8s and collects network traffic information in real-time to allocate computational resources proportionally to the distribution of network traffic
	FScaler is a reinforcement learning agent to scale and schedule containers based on defined cost functions
edge	KubeEdge architecture includes a network protocol stack called KubeBus, a distributed metadata store and synchronization service, and a lightweight agent (EdgeCore) for the edge
	The paper proposes a K8s compatible scheduler that integrates real-time information about edge infrastructure components enabling dynamic and advanced orchestration and management
	KEIDS is a K8s-Based Energy and Interference Driven Scheduler for container management in Industrial IoT applications
	The paper proposes a heuristic algorithm based on min–max to task offloading problems at the edge
	The paper proposes a framework for deploying an edge cloud of IoT applications based on the K8s orchestrator and microservice manager
	KubeHICE extends K8s to orchestrate heterogeneous-ISA architectures and provides performance-aware scheduling in the edge

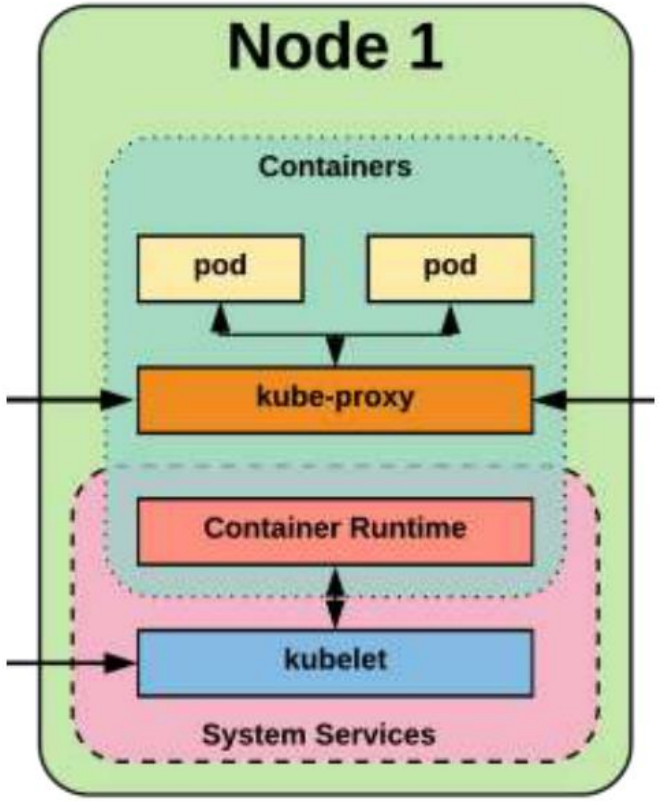

Master node
controller-manager
scheduler
API server
etcd
"please run these pods"
"what pods should i be running?"
Worker node
kubelet

Overall Architecture



Node-Components

- kubelet
- kube-proxy
- Container Runtime Engine



kubelet

- An agent that runs on each node in the cluster. It makes sure that containers are running in a pod.
- The kubelet takes a set of PodSpecs that are provided through various mechanisms and ensures that the containers described in those PodSpecs are running and healthy.

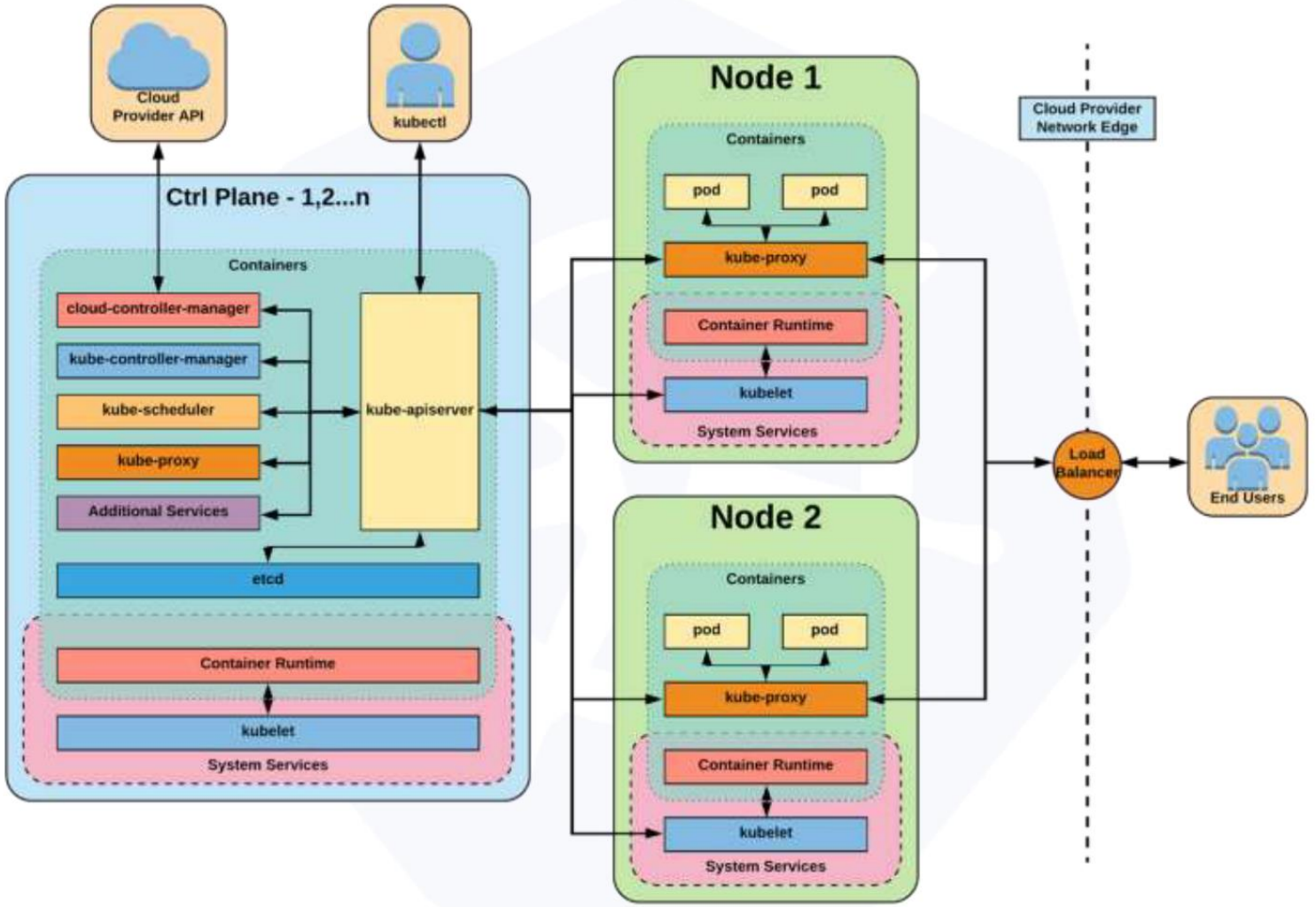
Kube-proxy

- Manages the network rules on each node.
- Performs connection forwarding or load balancing for Kubernetes cluster services.

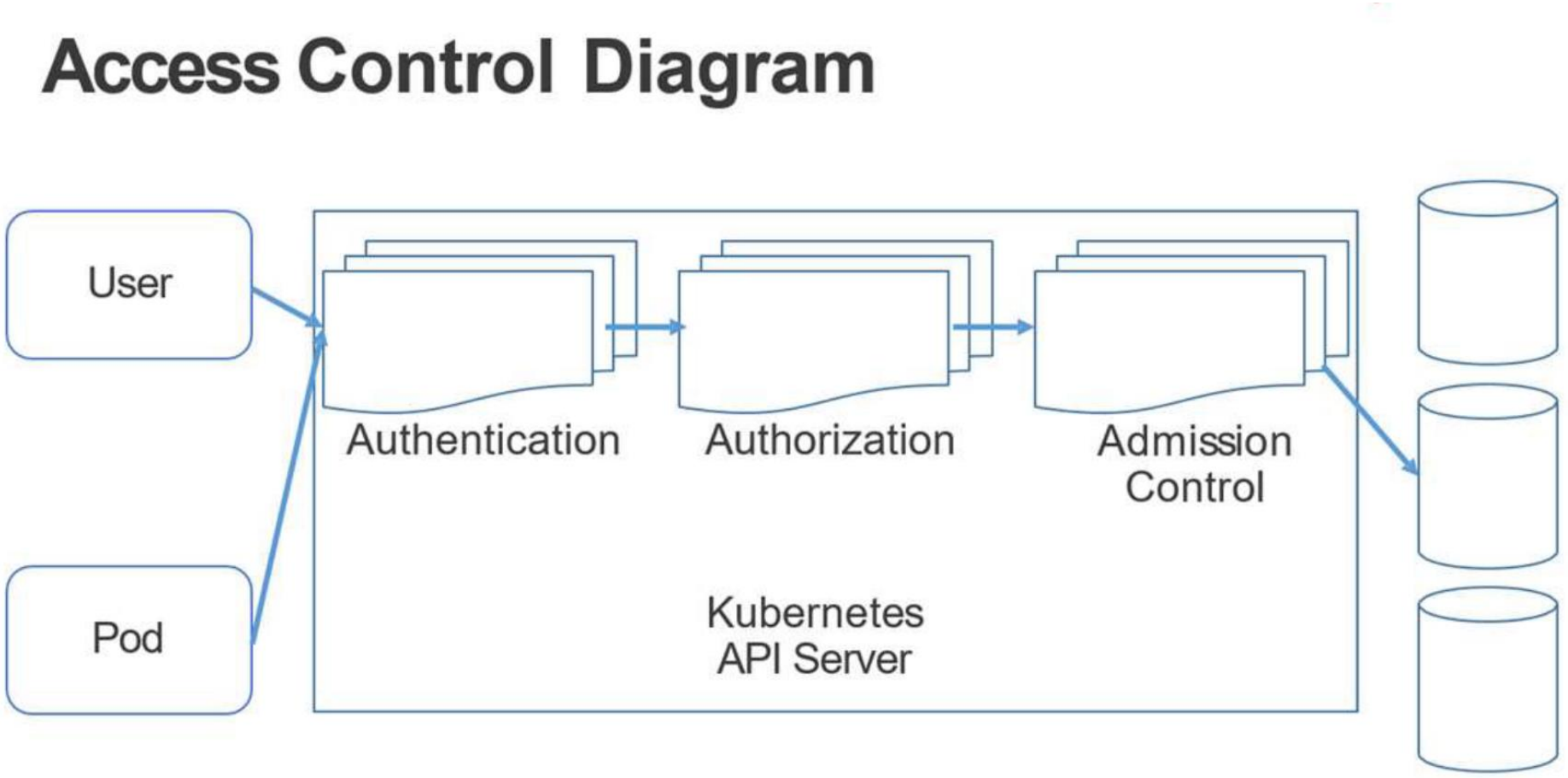
Container Runtime Engine

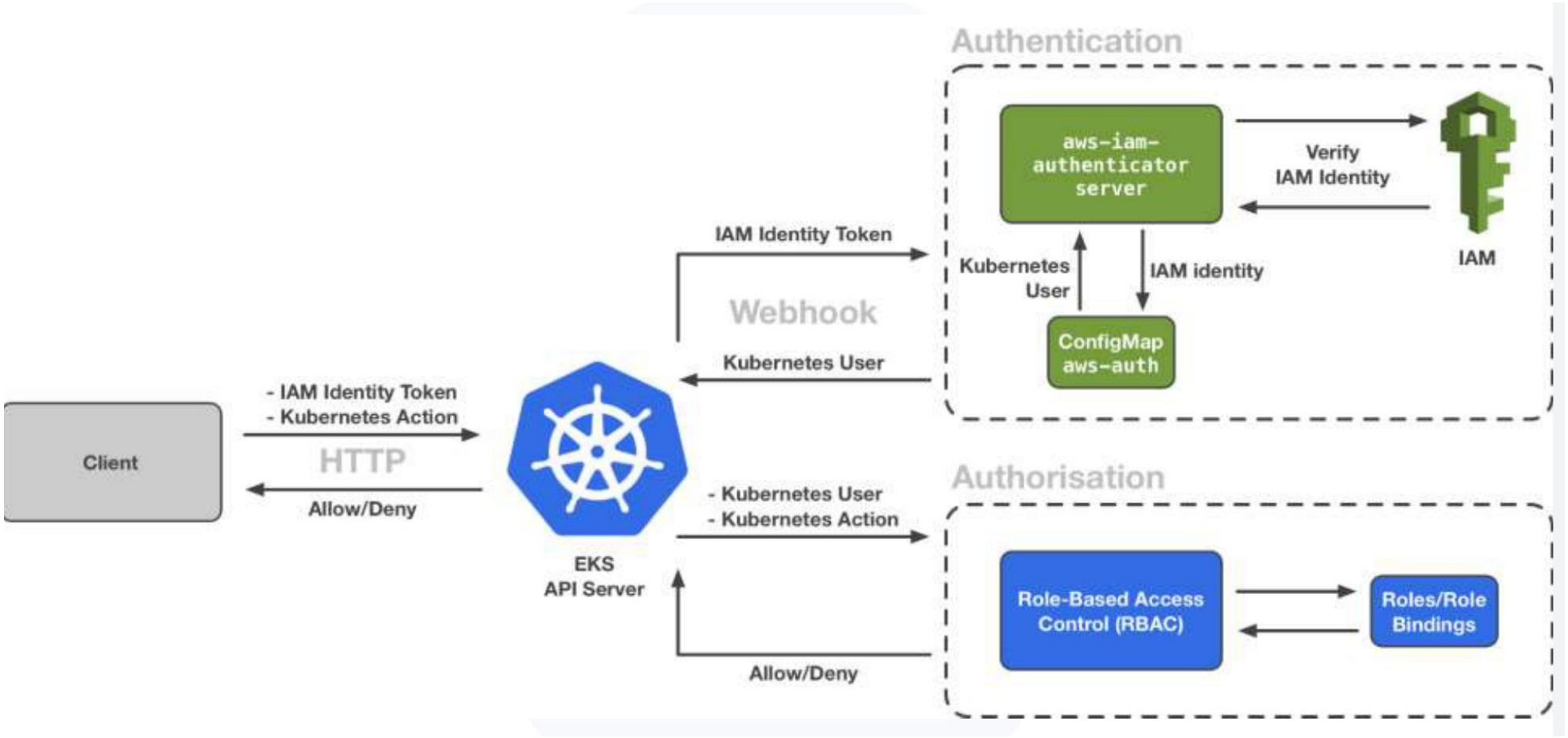
- A container runtime is a CRI (Container Runtime Interface) compatible application that executes and manages containers.
  - Containerd (docker)
  - Cri-o
  - Rkt
  - Kata (formerly clear and hyper)
  - Virtlet (VM CRI compatible runtime)

Overall Environment



Kubernetes


Access Control Diagram
User
Pod
Authentication
Authorization
Admission Control
Kubernetes API Server


Authentication
aws-iam-authenticator server
Verify IAM Identity
IAM
IAM Identity Token
Webhook
Kubernetes User
Kubernetes User
IAM identity
ConfigMap aws-auth
- IAM Identity Token
- Kubernetes Action
HTTP
Allow/Deny
Client
EKS API Server
Authorisation
- Kubernetes User
- Kubernetes Action
Role-Based Access Control (RBAC)
Roles/Role Bindings
Allow/Deny

Networking Rules

- All containers within a pod can communicate with each other unimpeded.
- All Pods can communicate with all other Pods without NAT.
- All nodes can communicate with all Pods (and vice-versa) without NAT.
- The IP that a Pod sees itself as is the same IP that others see it as.

Networking Rules

- **Container-to-Container**
  - Containers within a pod exist within the **same network namespace** and share an IP.
  - Enables intrapod communication over *localhost*.
- **Pod-to-Pod**
  - Allocated **cluster unique IP** for the duration of its life cycle.
  - Pods themselves are fundamentally ephemeral.

- **Pod-to-Service**
  - managed by **kube-proxy** and given a **persistent cluster unique IP**
  - exists beyond a Pod's lifecycle.
- **External-to-Service**
  - Handled by **kube-proxy.**
  - Works in cooperation with a cloud provider or other external entity (load balancer).

Core Objects and API

- Namespaces
- Pods
- Labels
- Selectors
- Services

Namespace

Namespaces are a logical cluster or environment, and are the primary method of partitioning a cluster or scoping access.

```
apiVersion: v1
kind: Namespace
metadata:
  name: prod
  labels:
    app: MyBigWebApp
```

```
$ kubectl get ns --show-labels
NAME          STATUS    AGE       LABELS
default       Active    11h       <none>
kube-public   Active    11h       <none>
kube-system   Active    11h       <none>
prod          Active    6s        app=MyBigWebApp
```

Pod Examples

```
apiVersion: v1
kind: Pod
metadata:
  name: pod-example
spec:
  containers:
  - name: nginx
    image: nginx:stable-alpine
    ports:
    - containerPort: 80
```

```
apiVersion: v1
kind: Pod
metadata:
  name: pod-example
  labels:
    app: nginx
spec:
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx
```

Key Container Attributes

- name - The name of the container
- image - The container image
- ports - array of ports to expose. Can be granted a friendly name and protocol may be specified
- env - array of environment variables
- command - Entrypoint array (equiv to Docker ENTRYPOINT)
- args - Arguments to pass to the command (equiv to Docker CMD)

Container

```
name: nginx
image: nginx:stable-alpine
ports:
  - containerPort: 80
    name: http
    protocol: TCP
env:
  - name: MYVAR
    value: isAwesome
command: ["/bin/sh", "-c"]
args: ["echo ${MYVAR}"]
```

Pod Attributes

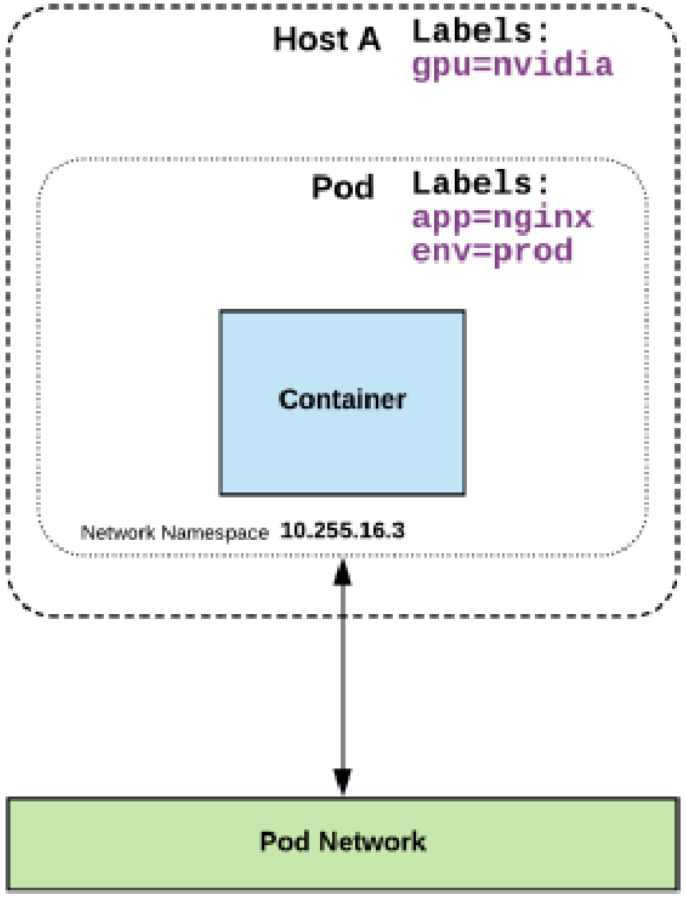
- Workload Controllers manage instances of Pods based off a provided template.
- Pod Templates are Pod specs with limited metadata.
- Controllers use Pod Templates to make actual pods.

```
apiVersion: v1
kind: Pod
metadata:
  name: pod-example
  labels:
    app: nginx
spec:
```

```
template:
  metadata:
    labels:
      app: nginx
  spec:
    containers:
    - name: nginx
      image: nginx
```

labels

- key-value pairs that are used to identify, describe and group together related sets of objects or resources.
- **NOT** characteristic of uniqueness.
- Have a strict syntax with a slightly limited character set*.



Resource Model

- **Request**: amount of a resource allowed to be used, with a strong guarantee of availability
  - CPU (seconds/second), RAM (bytes)
  - Scheduler will not over-commit requests
- **Limit**: max amount of a resource that can be used, regardless of guarantees
  - scheduler **ignores** limits

```
apiVersion: v1
kind: Pod
metadata:
  name: frontend
spec:
  containers:
  - name: db
    image: mysql
    resources:
      requests:
        memory: "64Mi"
        cpu: "250m"
      limits:
        memory: "128Mi"
        cpu: "500m"
```

- Mapping to Docker
  - —cpu-shares=requests.cpu
  - —cpu-quota=limits.cpu
  - —cpu-period=100ms
  - —memory=limits.memory

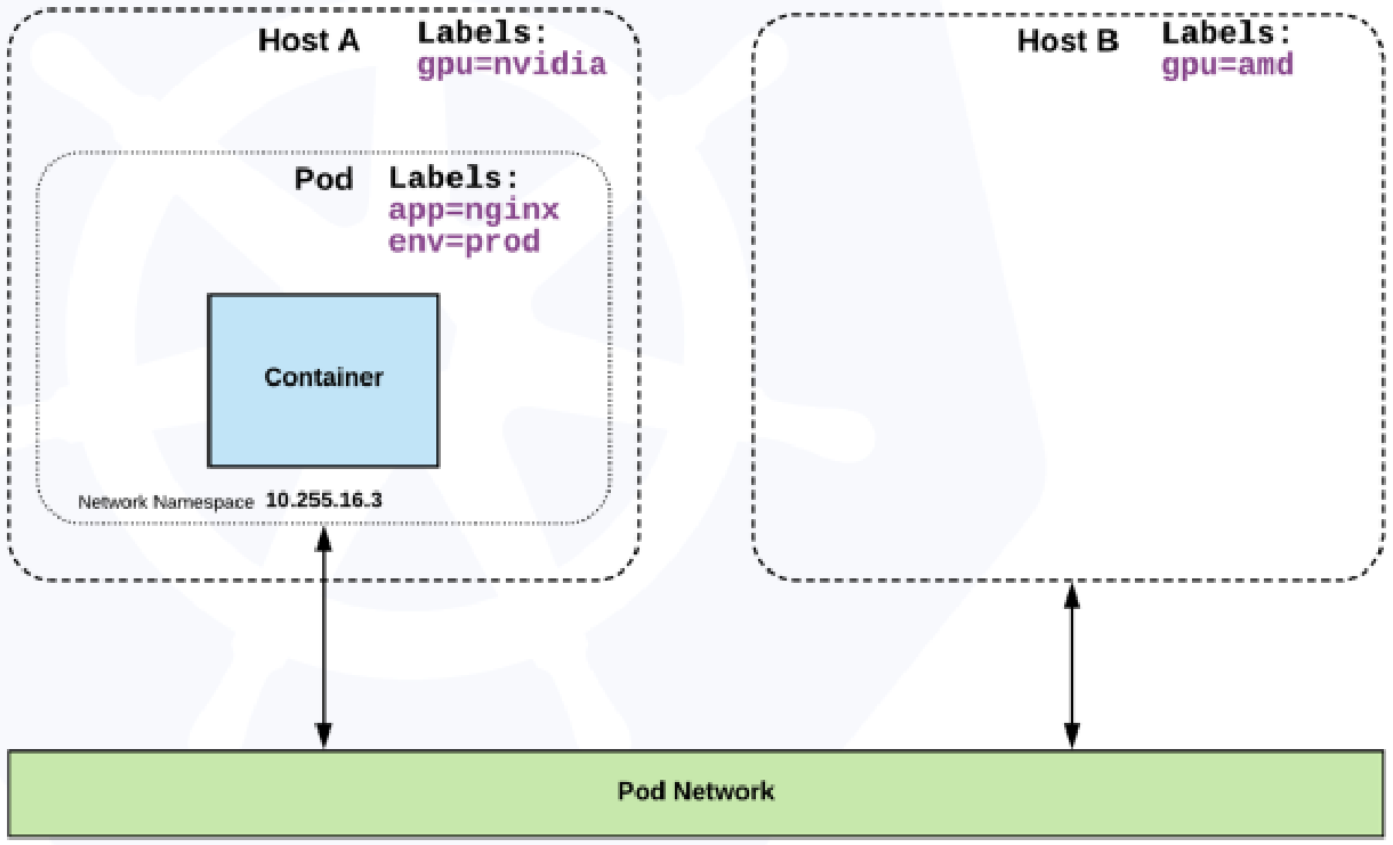
Selectors

Selectors use labels to filter or select objects, and are used throughout Kubernetes.

```
apiVersion: v1
kind: Pod
metadata:
  name: pod-label-example
  labels:
    app: nginx
    env: prod
spec:
  containers:
  - name: nginx
    image: nginx:stable-alpine
    ports:
    - containerPort: 80
  nodeSelector:
    gpu: nvidia
```

Selectors Example

```
apiVersion: v1
kind: Pod
metadata:
  name: pod-label-example
  labels:
    app: nginx
    env: prod
spec:
  containers:
  - name: nginx
    image: nginx:stable-alpine
    ports:
    - containerPort: 80
  nodeSelector:
    gpu: nvidia
```



Selector Types

Equality based selectors allow for simple filtering (=,==, or !=).

```
selector:
  matchLabels:
    gpu: nvidia
```

Set-based selectors are supported on a limited subset of objects. However, they provide a method of filtering on a set of values, and supports multiple operators including: in, notin, and exist.

```
selector:
  matchExpressions:
    - key: gpu
      operator: in
      values: [“nvidia”]
```

Services

- **Unified method of accessing** the exposed workloads of Pods.
- **Durable resource** (unlike Pods)
  - static cluster-unique IP
  - static namespaced DNS name

<service name>.<namespace>.svc.cluster.local

ClusterIP Service

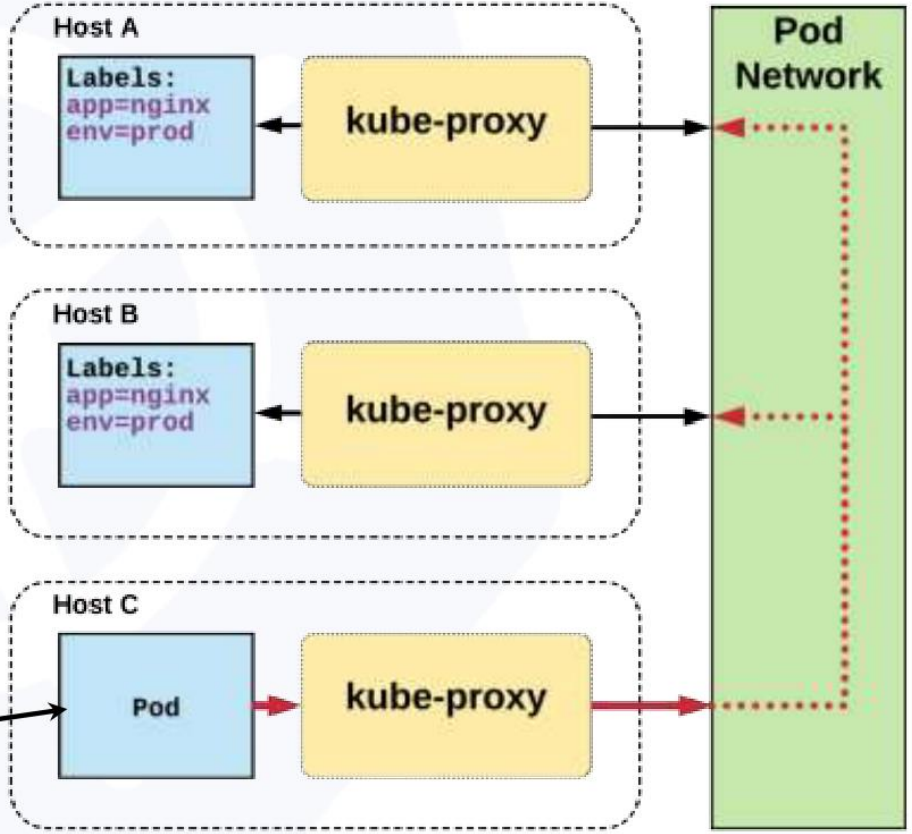
ClusterIP services exposes a service on a strictly cluster internal virtual IP.

```
apiVersion: v1
kind: Service
metadata:
  name: example-prod
spec:
  selector:
    app: nginx
    env: prod
  ports:
  - protocol: TCP
    port: 80
    targetPort: 80
```

ClusterIP Service

```
Name:         example-prod
Selector:     app=nginx,env=prod
Type:         ClusterIP
IP:           10.96.28.176
Port:         <unset>  80/TCP
TargetPort:   80/TCP
Endpoints:    10.255.16.3:80,
              10.255.16.4:80
```

```
/ # nslookup example-prod.default.svc.cluster.local

Name:      example-prod.default.svc.cluster.local
Address 1: 10.96.28.176 example-prod.default.svc.cluster.local
```

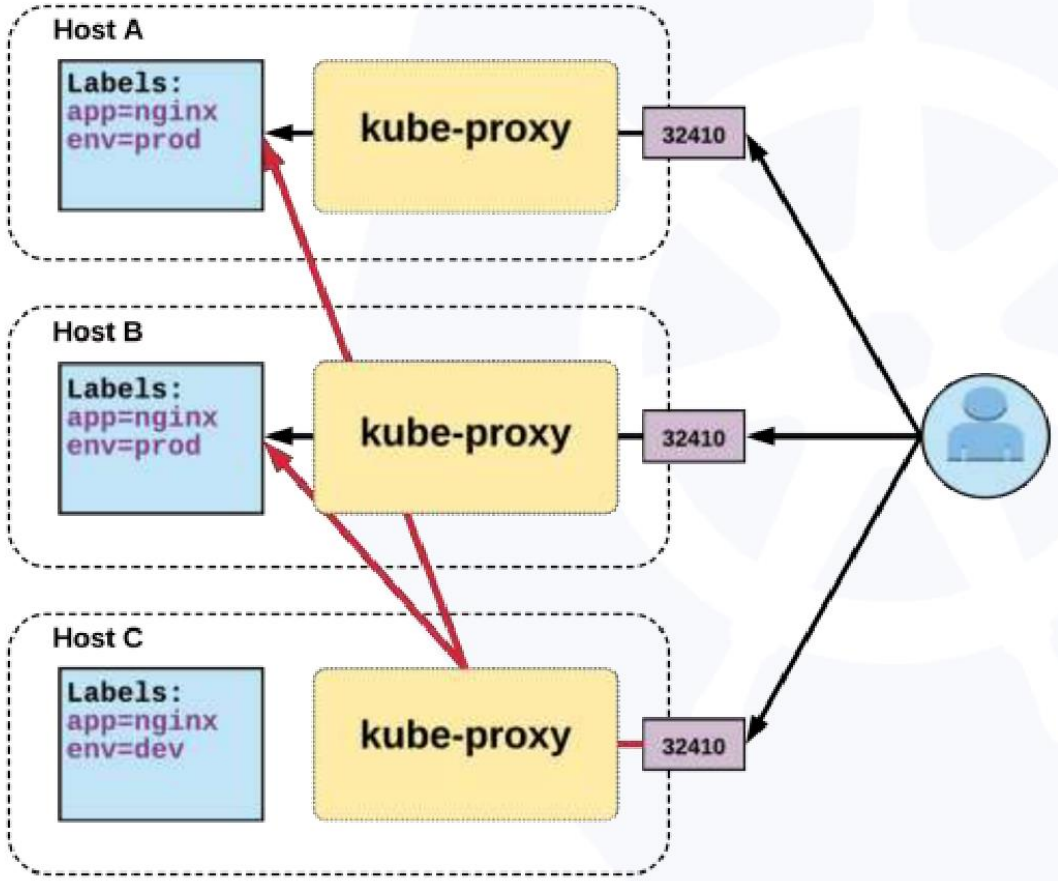


NodePort Service

- **NodePort** services extend the **ClusterIP** service.
- Exposes a port on every node's IP.
- Port can either be statically defined, or dynamically taken from a range between 30000-32767.

```
apiVersion: v1
kind: Service
metadata:
  name: example-prod
spec:
  type: NodePort
  selector:
    app: nginx
    env: prod
  ports:
  - nodePort: 32410
    protocol: TCP
    port: 80
    targetPort: 80
```

NodePort Service



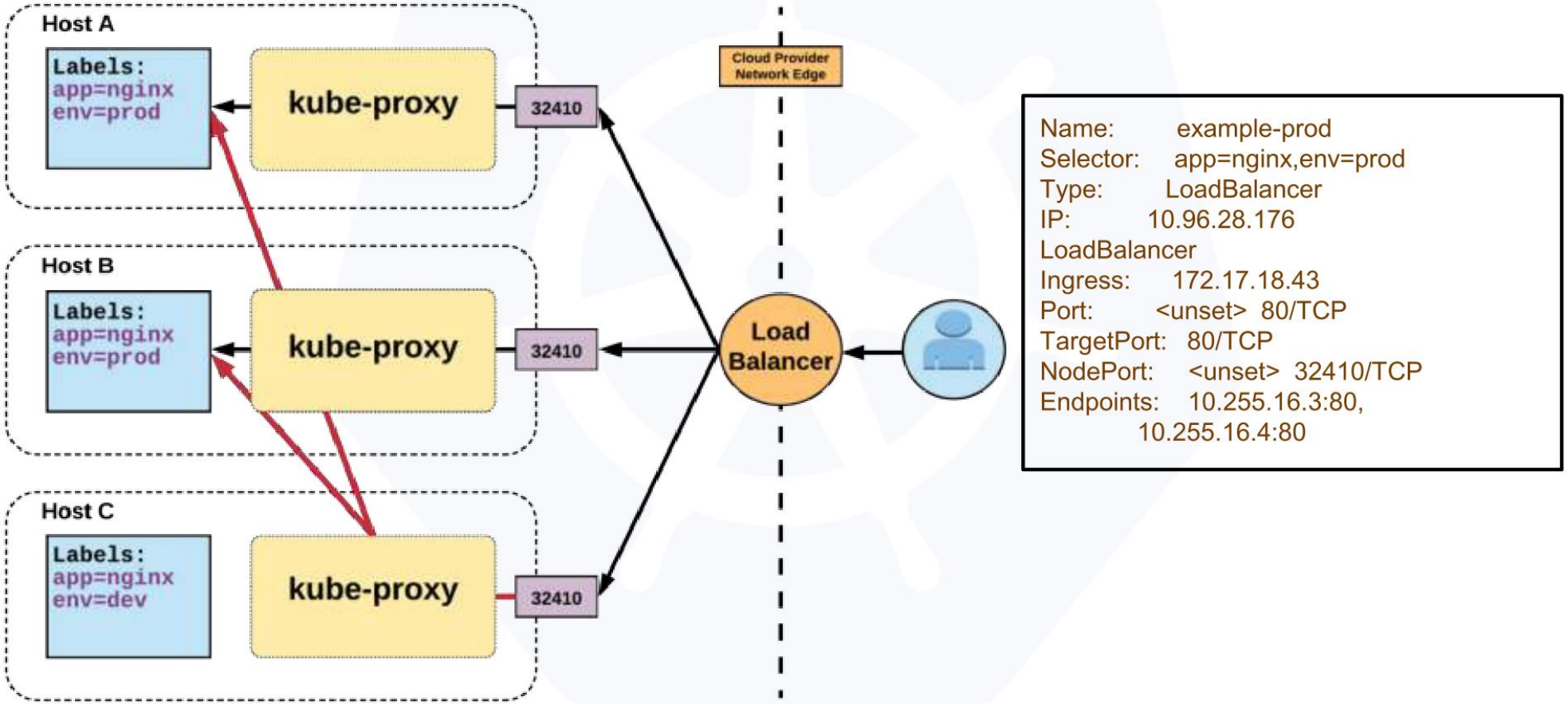

```
Name:         example-prod
Selector:     app=nginx,env=prod
Type:         NodePort
IP:           10.96.28.176
Port:         <unset>  80/TCP
TargetPort:   80/TCP
NodePort:     <unset>  32410/TCP
Endpoints:    10.255.16.3:80,
              10.255.16.4:80
```

LoadBalancer Service

- **LoadBalancer** services extend **NodePort.**
- Works in conjunction with an external system to map a cluster external IP to the exposed service.

```
apiVersion: v1
kind: Service
metadata:
  name: example-prod
spec:
  type: LoadBalancer
  selector:
    app: nginx
    env: prod
  ports:
    protocol: TCP
    port: 80
    targetPort: 80
```

LoadBalancer Service



ExternalName Service

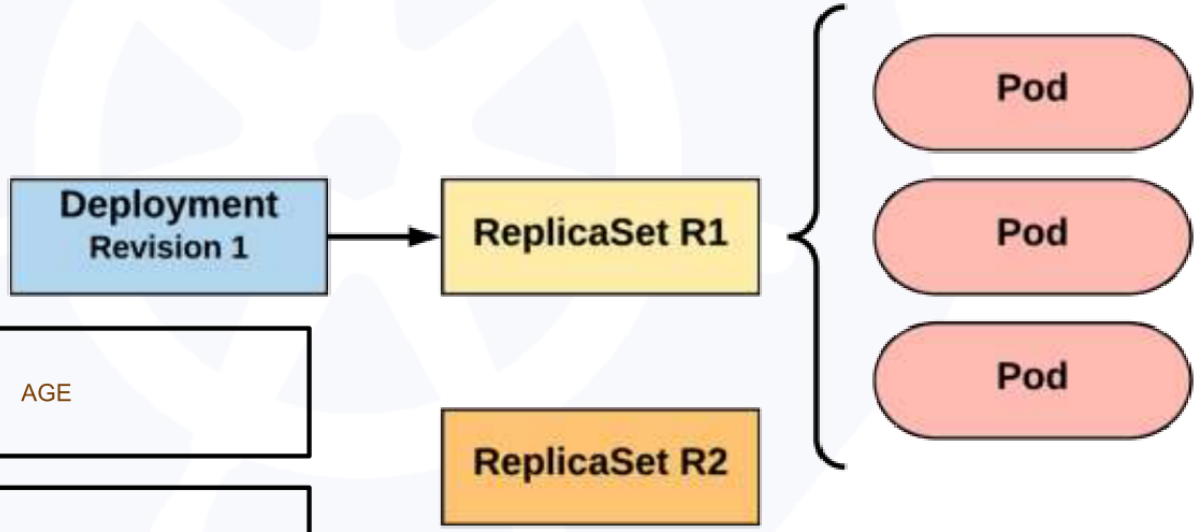
- **ExternalName** is used to reference endpoints **OUTSIDE** the cluster.
- Creates an internal **CNAME** DNS entry that aliases another.

```
apiVersion: v1
kind: Service
metadata:
  name: example-prod
spec:
  type: ExternalName
spec:
  externalName: example.com
```

Rolling Update

Updating pod template generates a new **ReplicaSet** revision.

R1 pod-template-hash:
676677fff
R2 pod-template-hash:
54f7ff7d6d



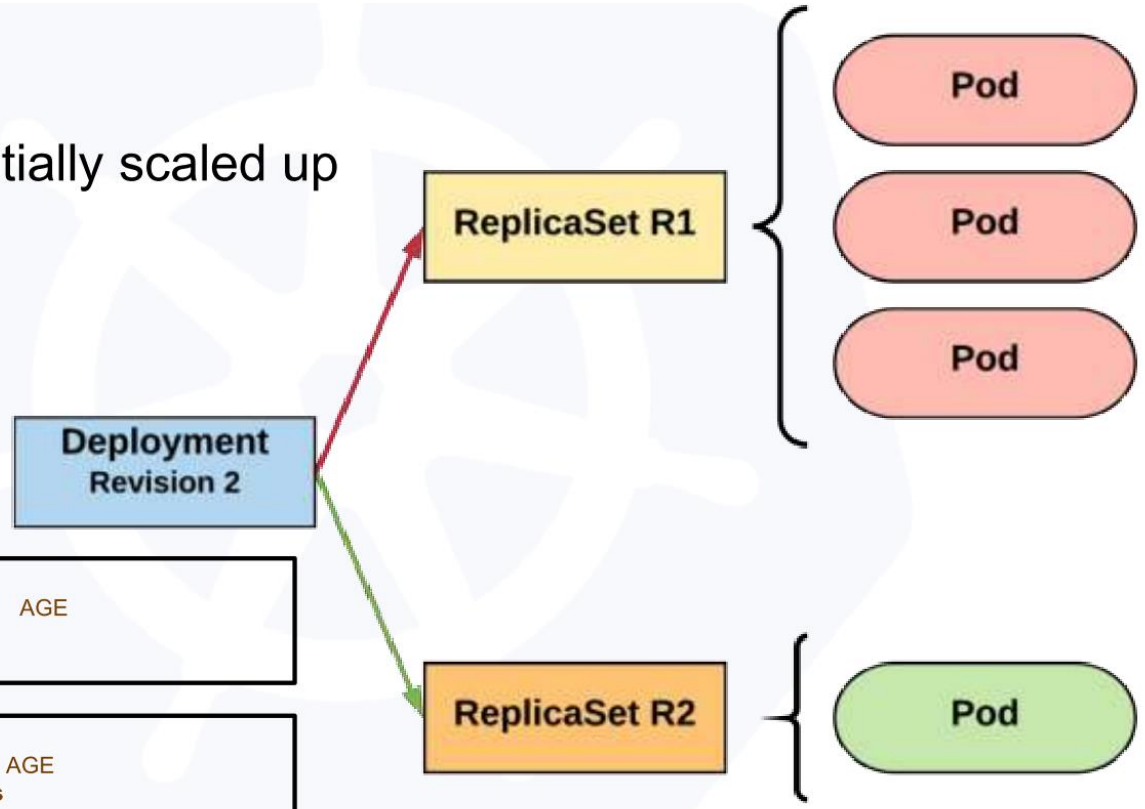

```
$ kubectl get replicaset
NAME                DESIRED   CURRENT   READY   AGE
mydep-6766777fff    3         3         3       5h
```

```
$ kubectl get pods
NAME                    READY   STATUS    RESTARTS   AGE
mydep-6766777fff-9r2zn  1/1     Running   0          5h
mydep-6766777fff-hsfz9  1/1     Running   0          5h
mydep-6766777fff-sjxhf  1/1     Running   0          5h
```

Rolling Update

New **ReplicaSet** is initially scaled up based on maxSurge.

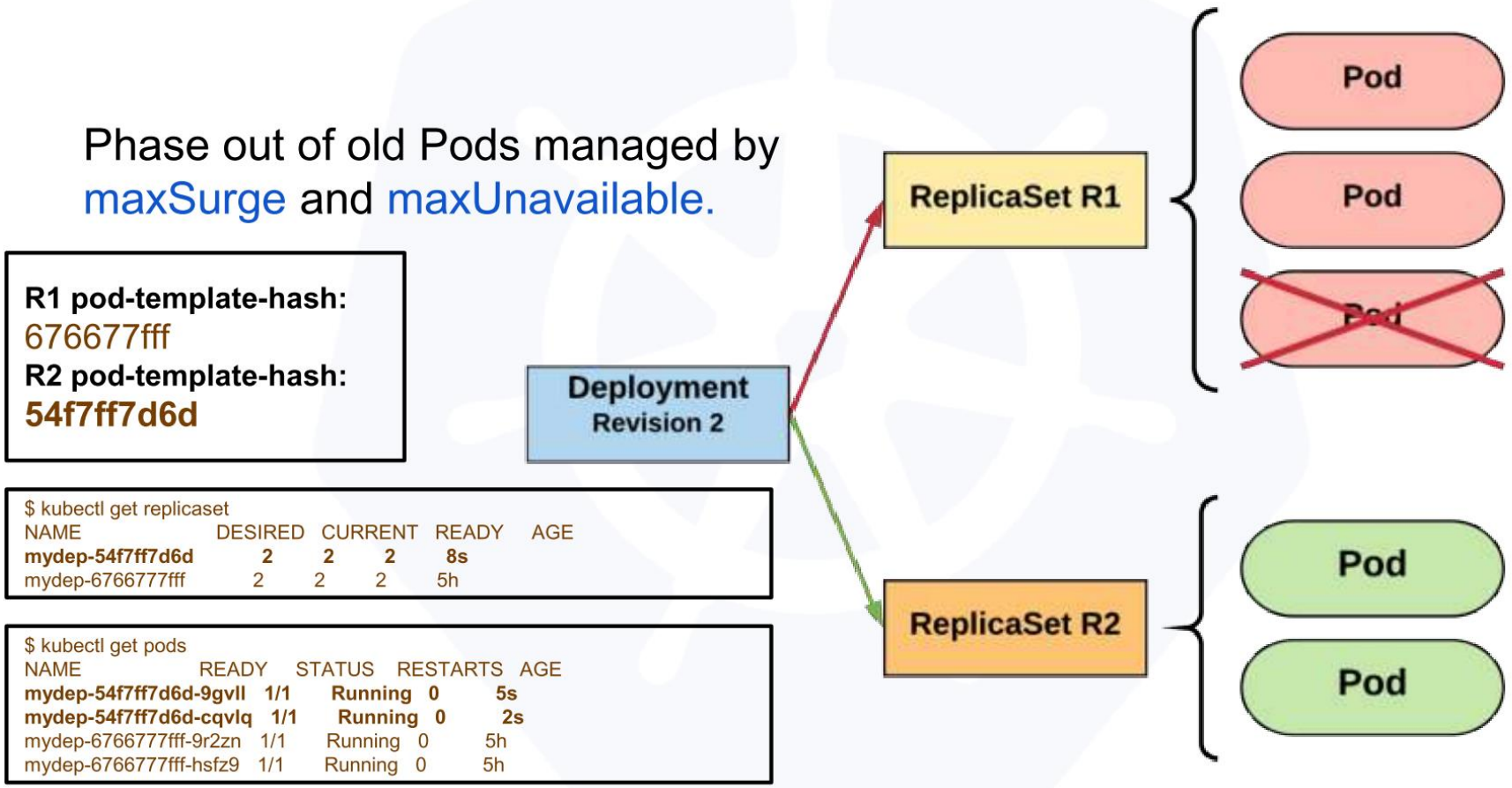
R1 pod-template-hash:
676677fff
R2 pod-template-hash:
54f7ff7d6d



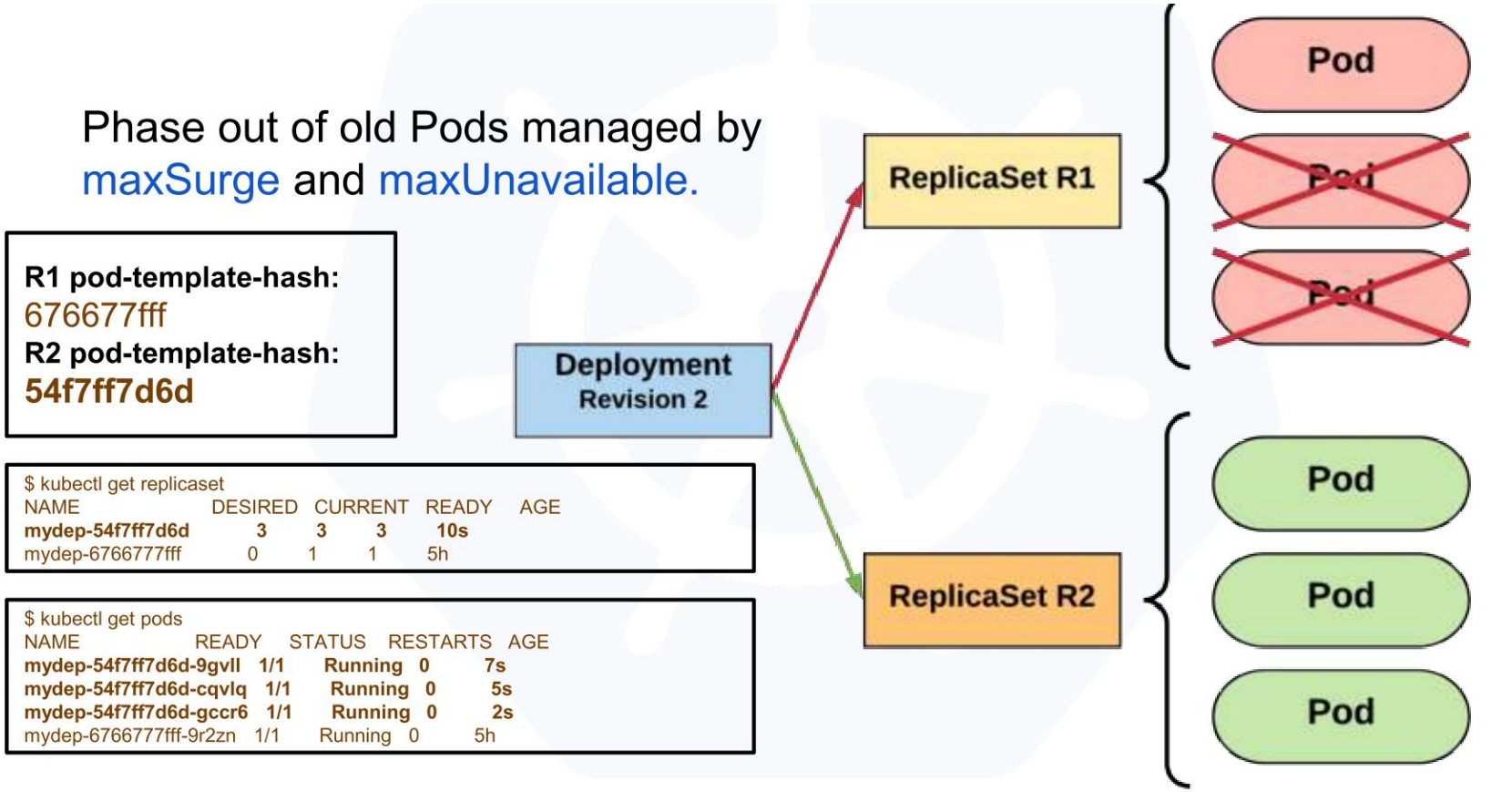

```
$ kubectl get replicaset
NAME                DESIRED   CURRENT   READY   AGE
mydep-54f7ff7d6d      1         1       1       5s
mydep-6766777fff      2         3       3       5h
```

```
$ kubectl get pods
NAME                      READY   STATUS    RESTARTS   AGE
mydep-54f7ff7d6d-9gvll    1/1     Running   0          2s
mydep-6766777fff-9r2zn    1/1     Running   0          5h
mydep-6766777fff-hsfz9    1/1     Running   0          5h
mydep-6766777fff-sjxhf    1/1     Running   0          5h
```

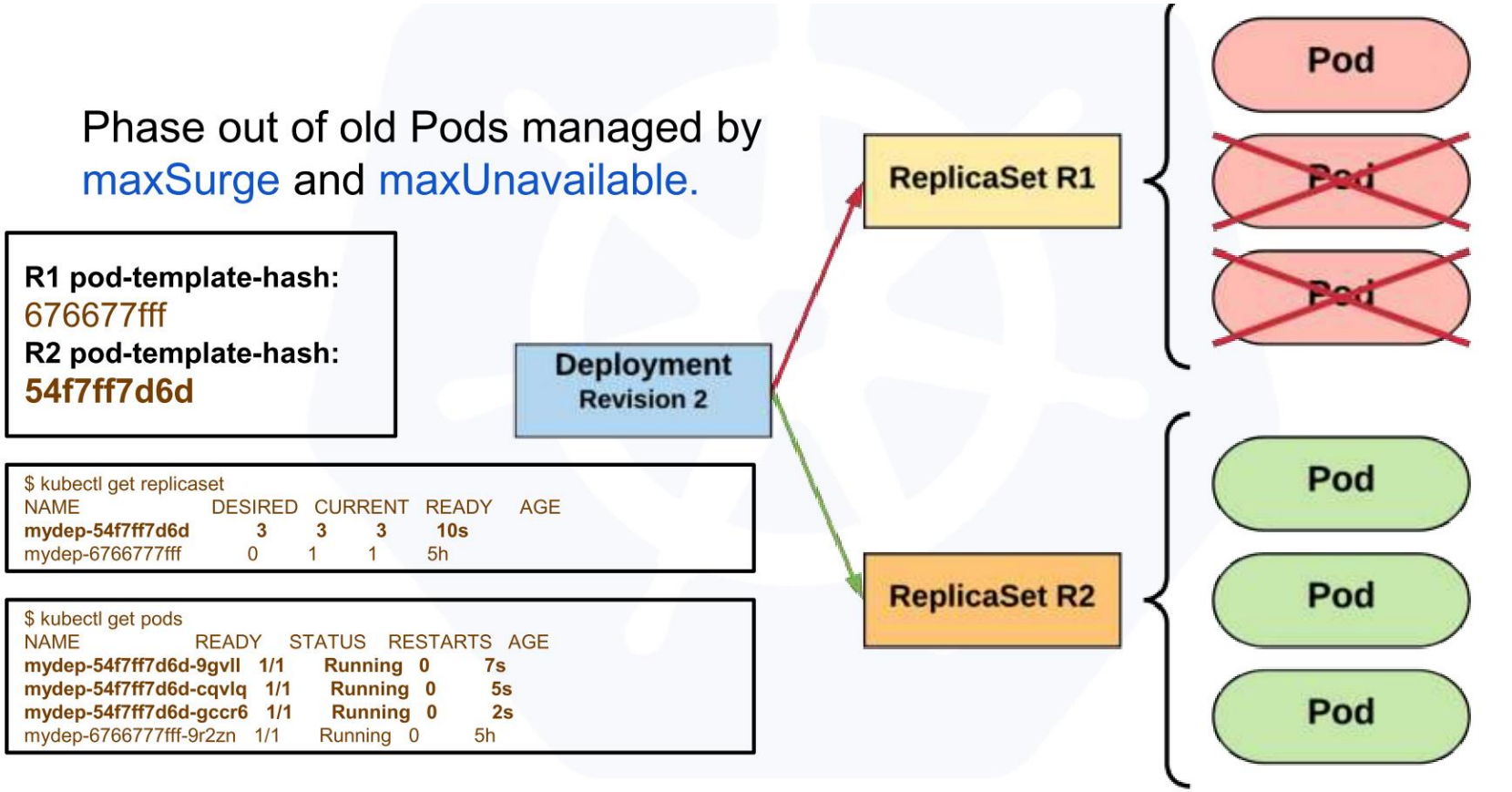
Rolling Update



Rolling Update



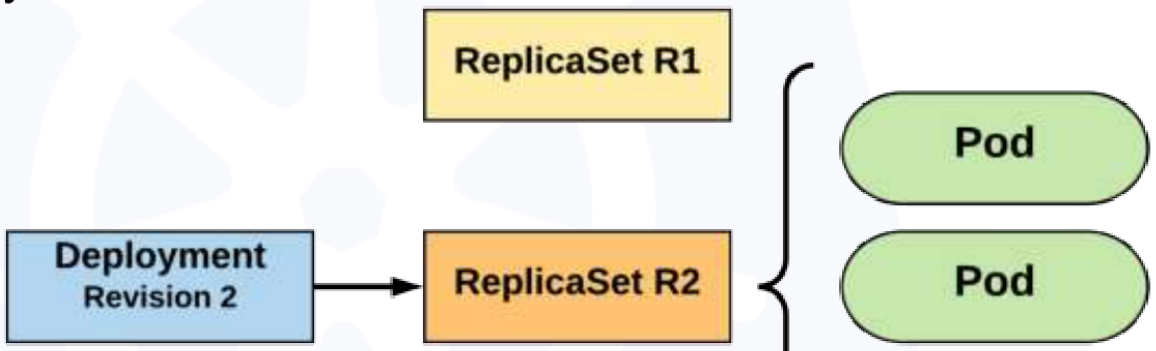
Rolling Update



Rolling Update

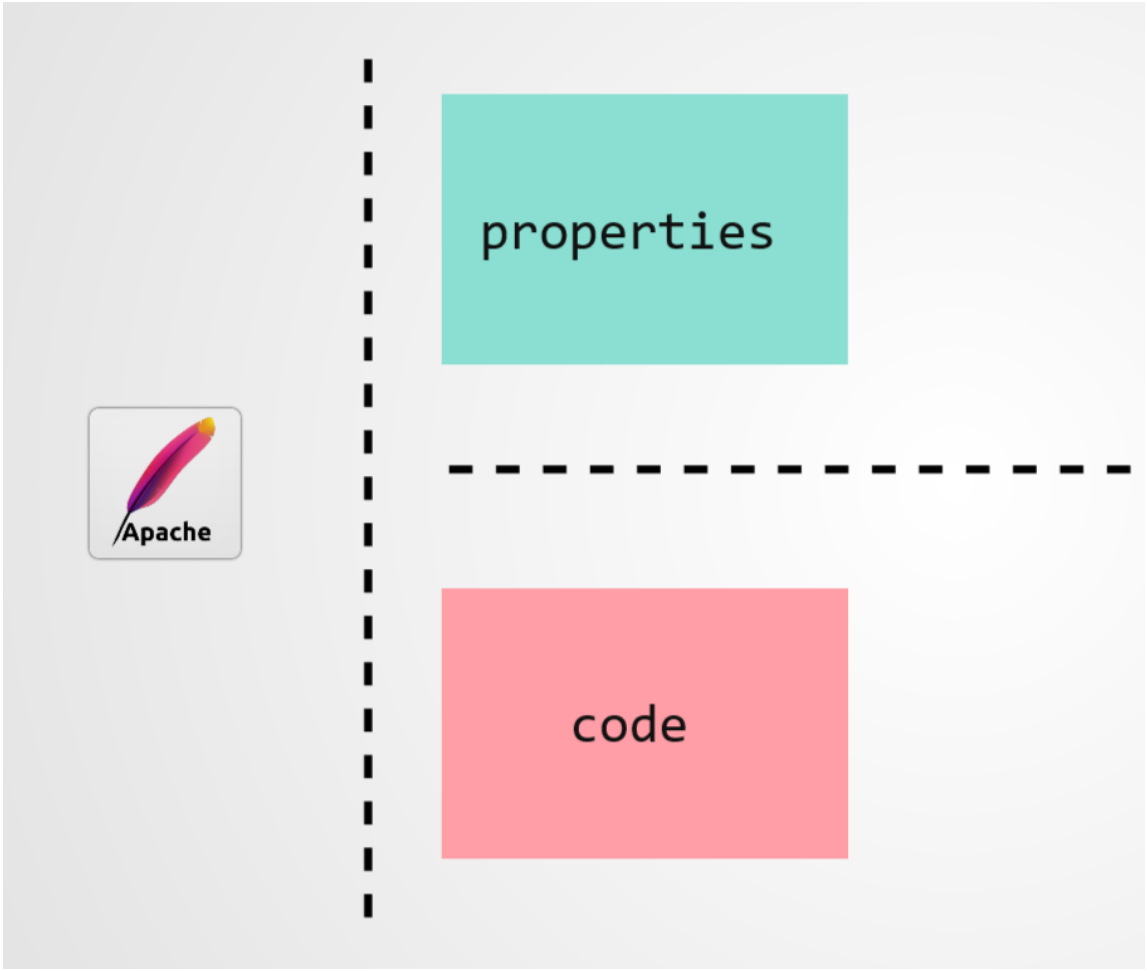
Updated to new deployment revision completed.

R1 pod-template-hash:
676677fff
R2 pod-template-hash:
54f7ff7d6d




```
$ kubectl get replicaset
NAME                DESIRED   CURRENT   READY     AGE
mydep-54f7ff7d6d    3         3         3         15s
mydep-6766777fff    0         0         0         5h
```

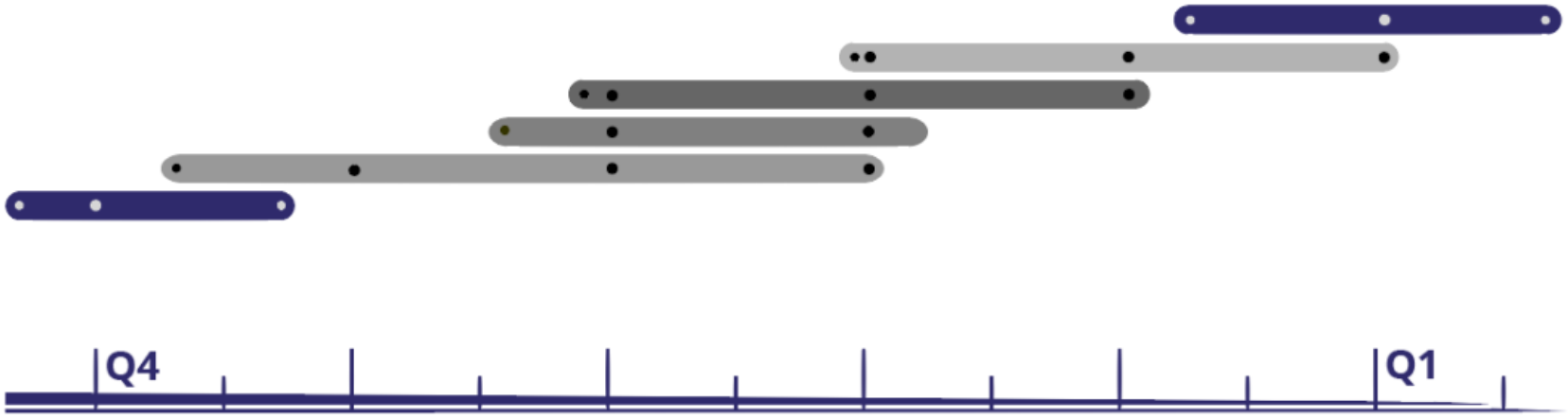
```
$ kubectl get pods
NAME                    READY     STATUS    RESTARTS   AGE
mydep-54f7ff7d6d-9gvll  1/1       Running   0          12s
mydep-54f7ff7d6d-cqvlq  1/1       Running   0          10s
mydep-54f7ff7d6d-gccr6  1/1       Running   0          7s
```


properties
Apache
code

Puppet and Chef

Timeline Reduction Through Automation

Reduce The Timeline



Motivation

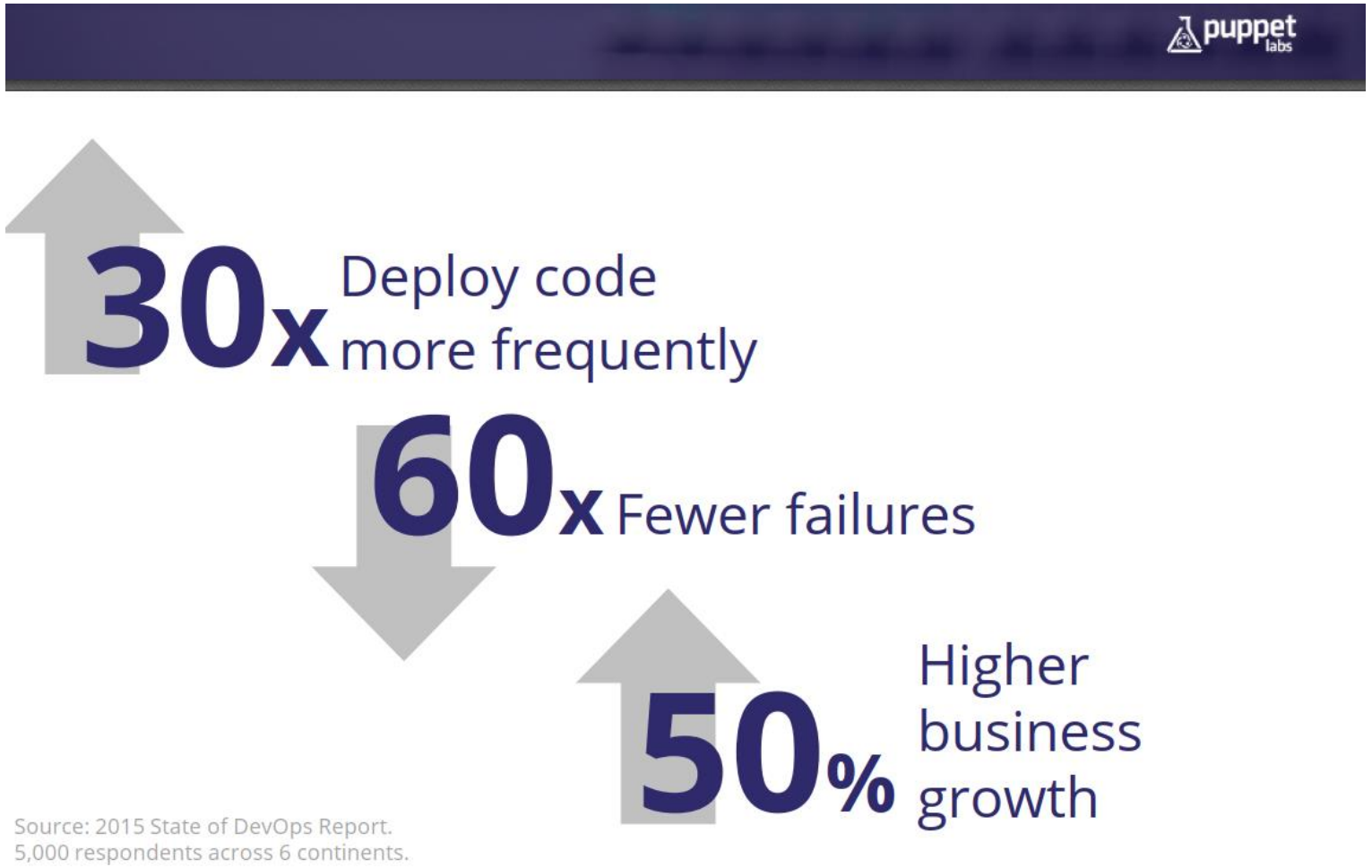
Common Challenges

- Too much fire fighting
- Slow deployments
- Scripting & manual processes aren't cutting it
- Difficult to keep up with demands from the business

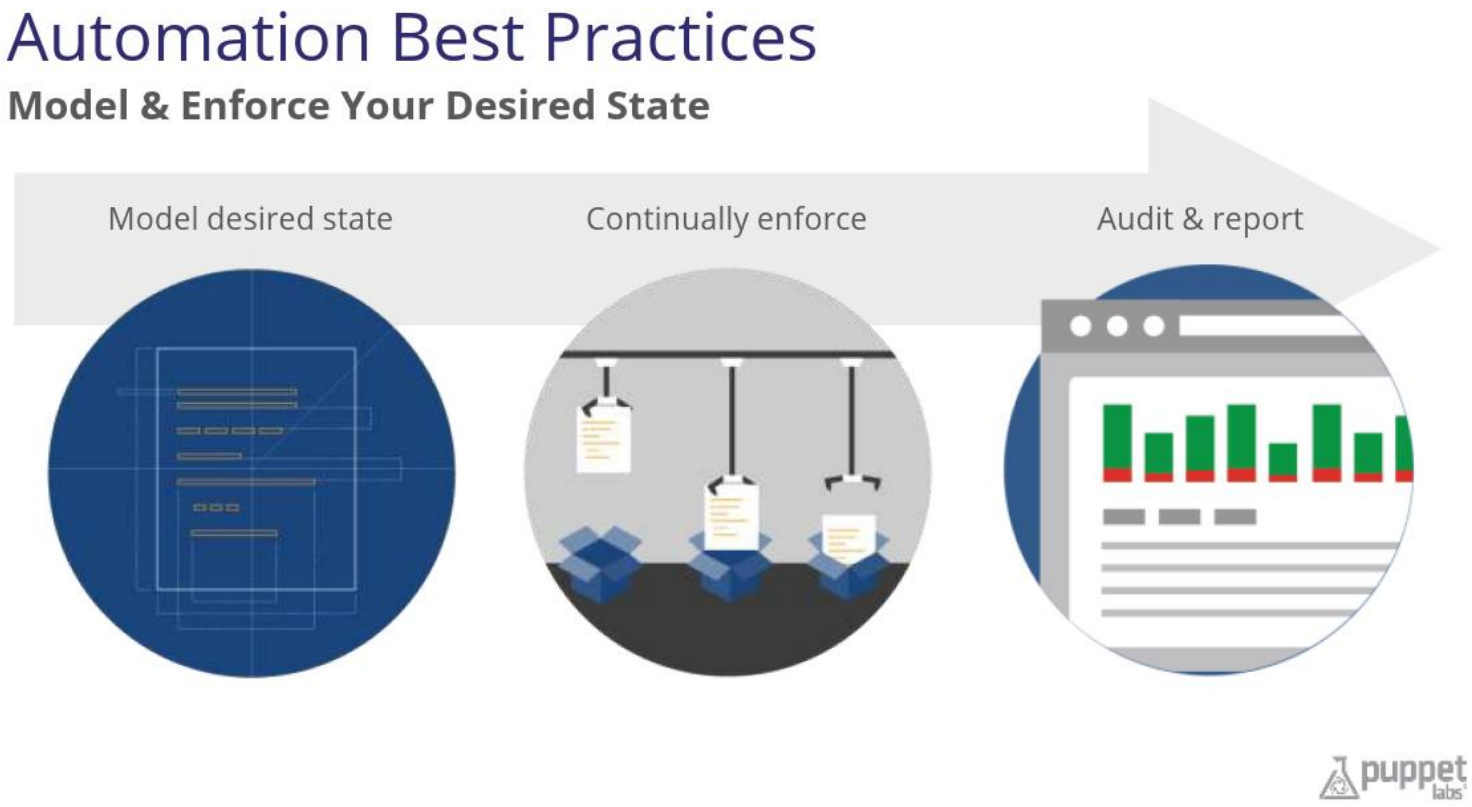
Key Initiatives

- Deliver value to business faster, more reliably
- Meet compliance requirements
- Adopt DevOps practices
- Adopt new technology while supporting & sun-setting old

From Puppet Labs



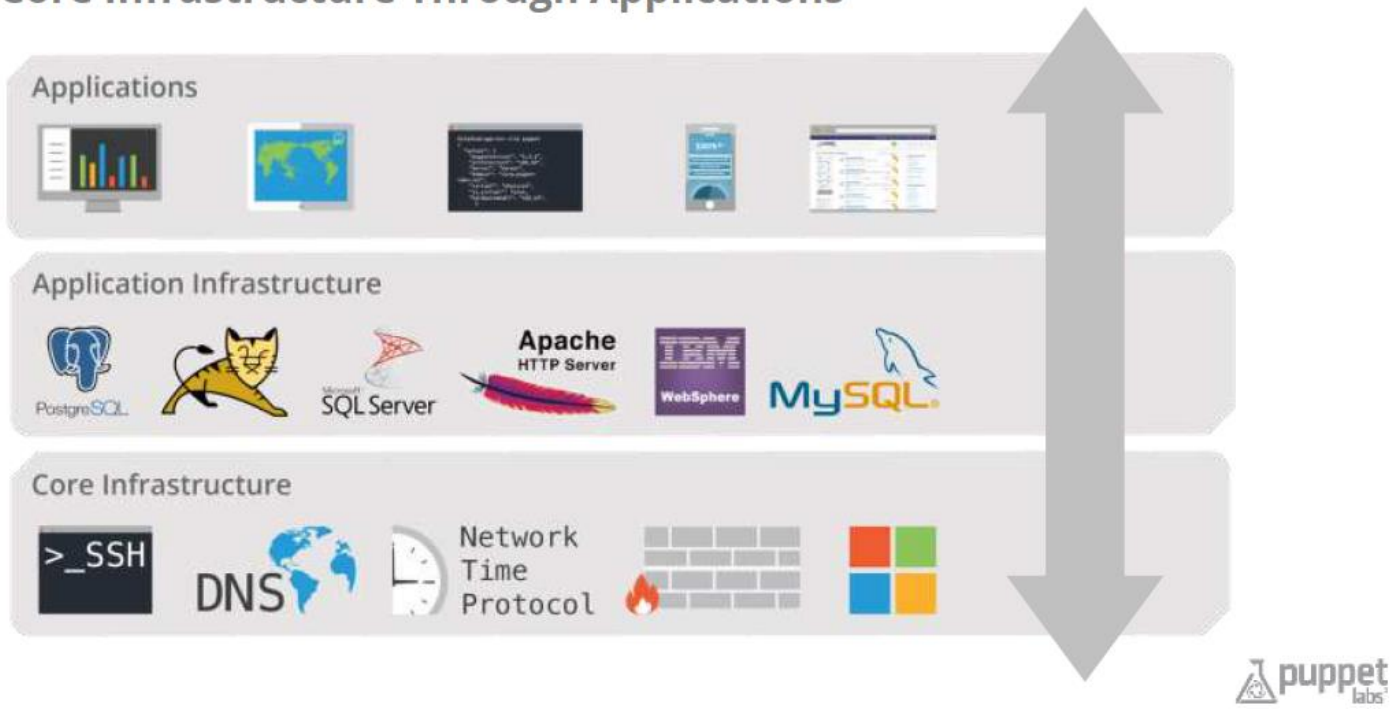
Puppet Approach - Declarative



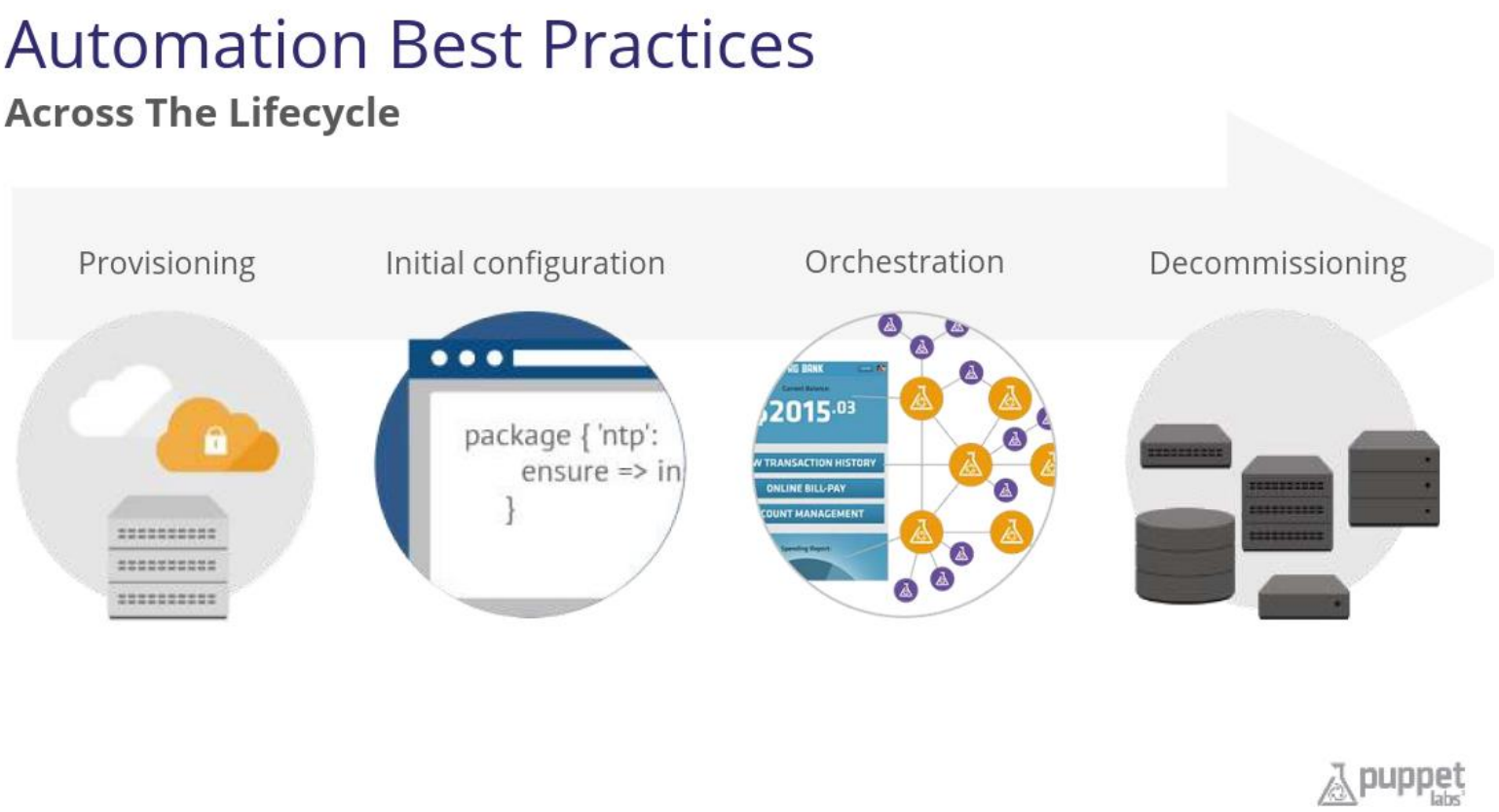
Puppet Approach

Automation Best Practices

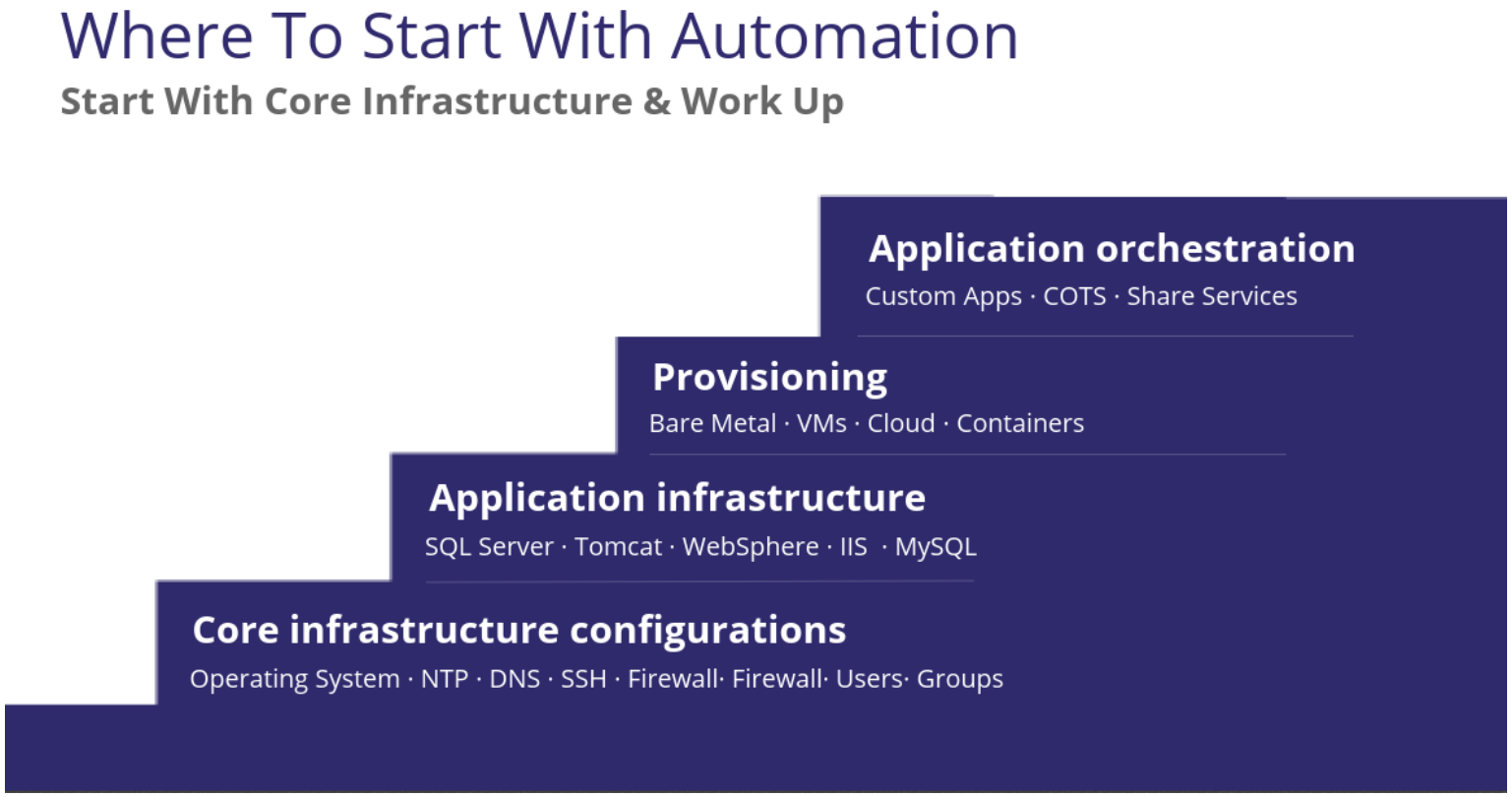
From Core Infrastructure Through Applications



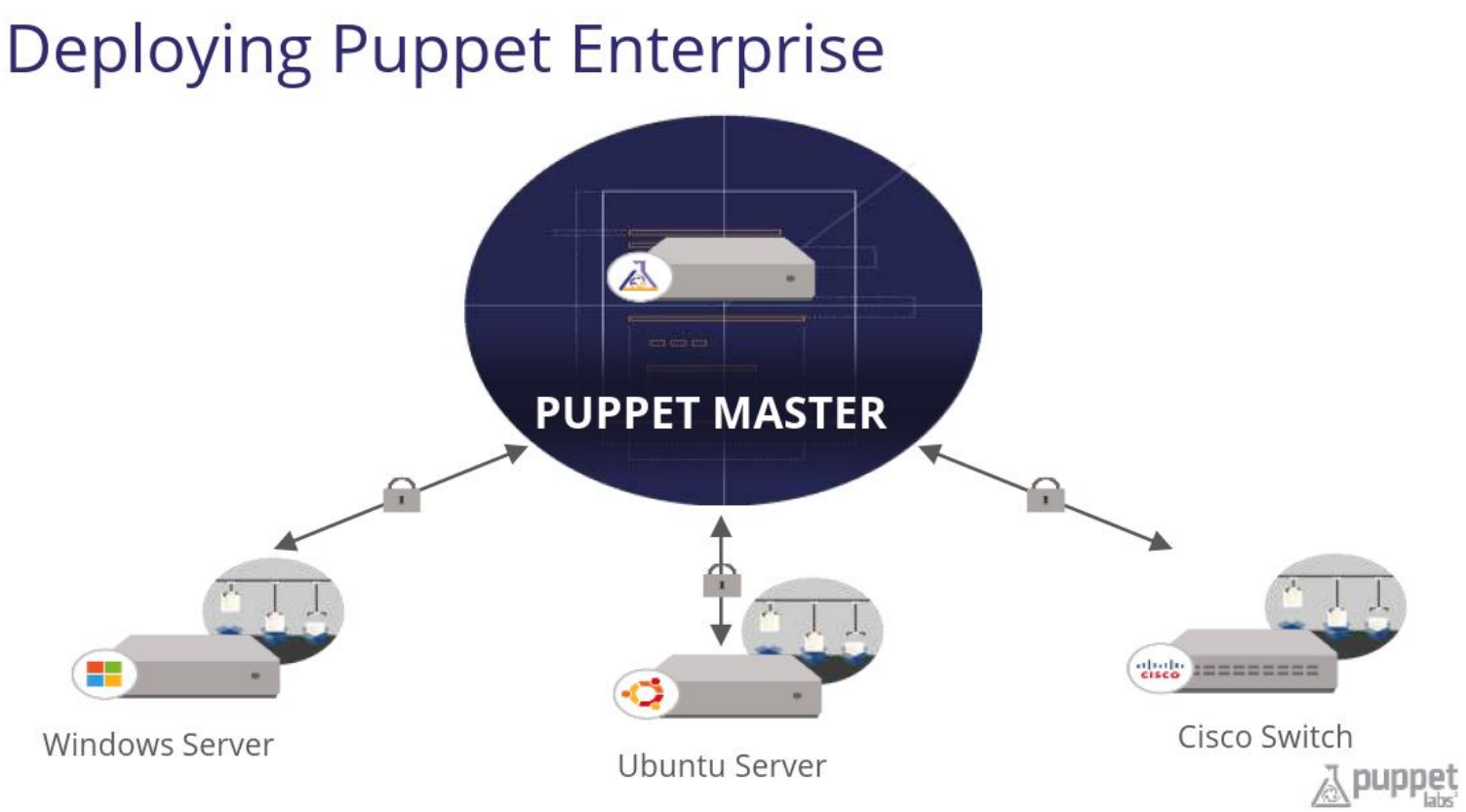
Puppet Approach



Puppet Approach

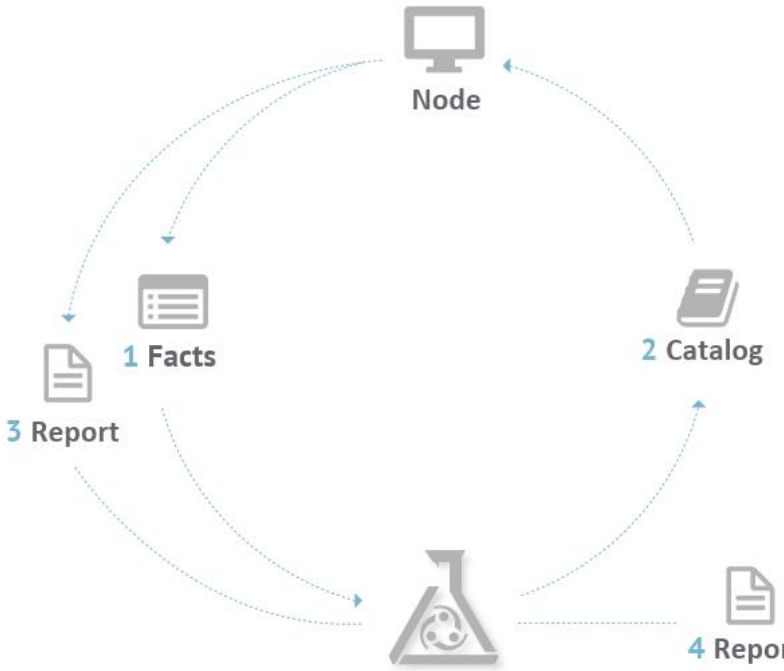


Puppet Approach



Puppet Approach

Lifecycle of a Puppet Run



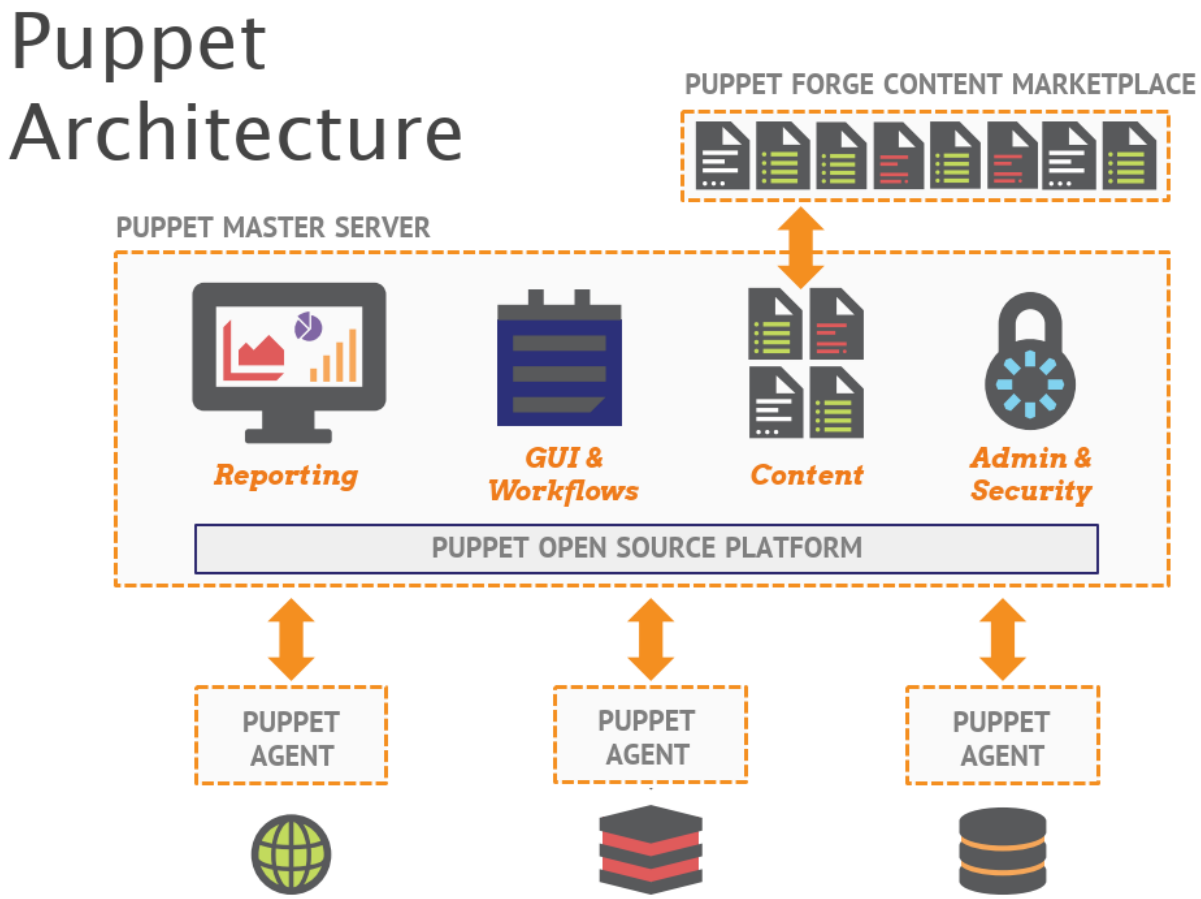
1. Facts
The node sends data about its state to the puppet master server.

2. Catalog
Puppet uses the facts to compile a Catalog that specifies how the node should be configured.

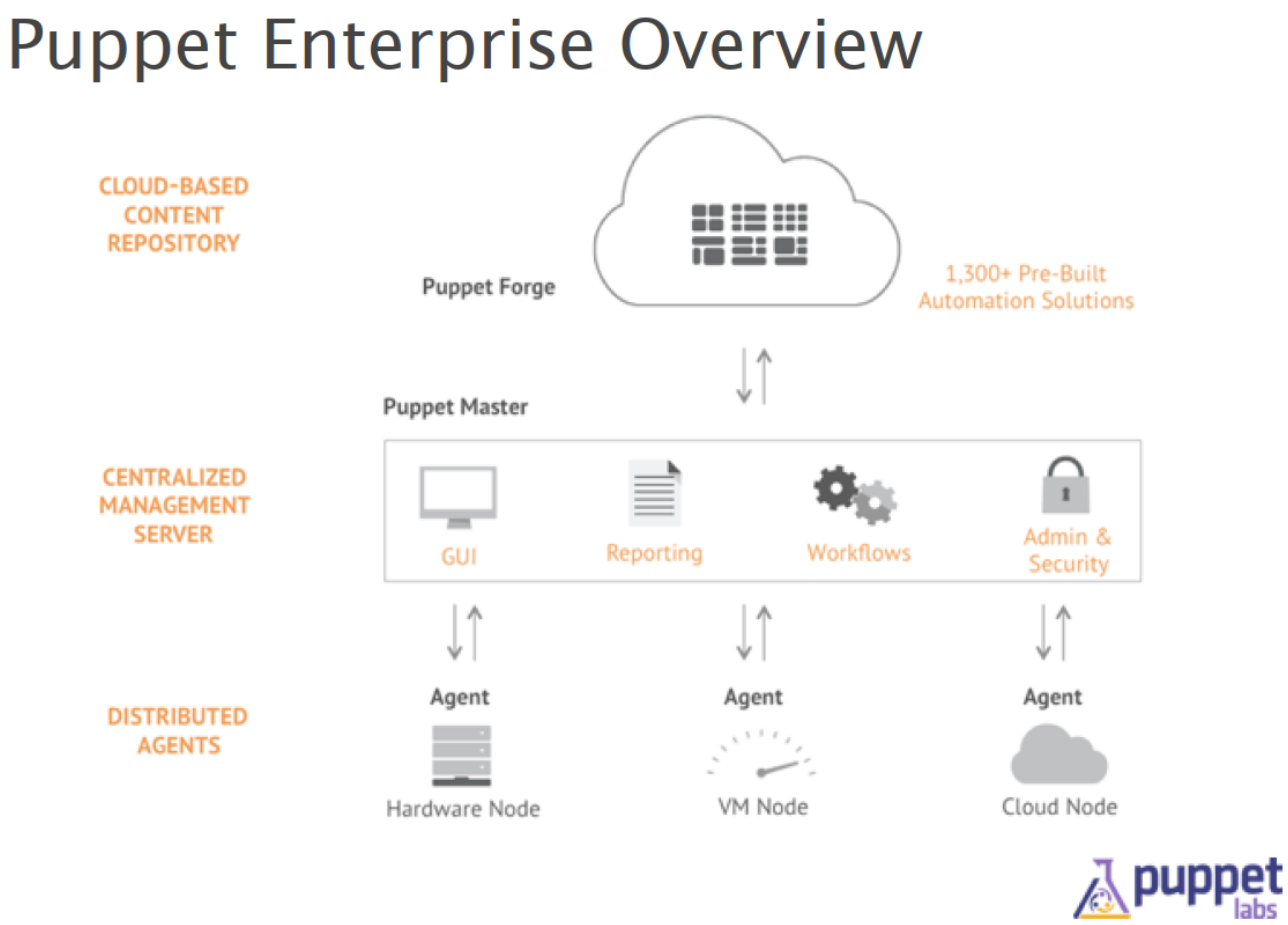
3. Report
Configuration changes are reported back to the Puppet Master.

4. Report
Puppet's open API can also send data to 3rd party tools.

Puppet Approach



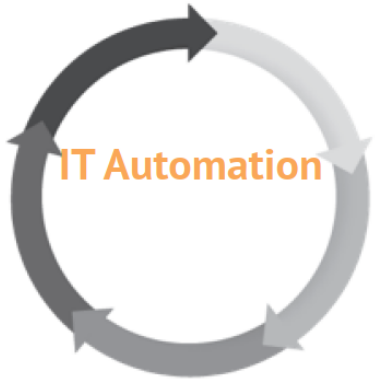
Puppet Approach



Puppet Approach

Puppet Enterprise

IT automation for end-to-end infrastructure lifecycle management



Discovery of nodes, resources, and status using real-time data

Provisioning of bare metal, virtual, and cloud capacity

Configuration installation and configuration of operating systems and applications and automated enforcement

Orchestration of multi-step operations to targeted collections of nodes

Reporting of all state changes of all resources across all nodes

Puppet Approach

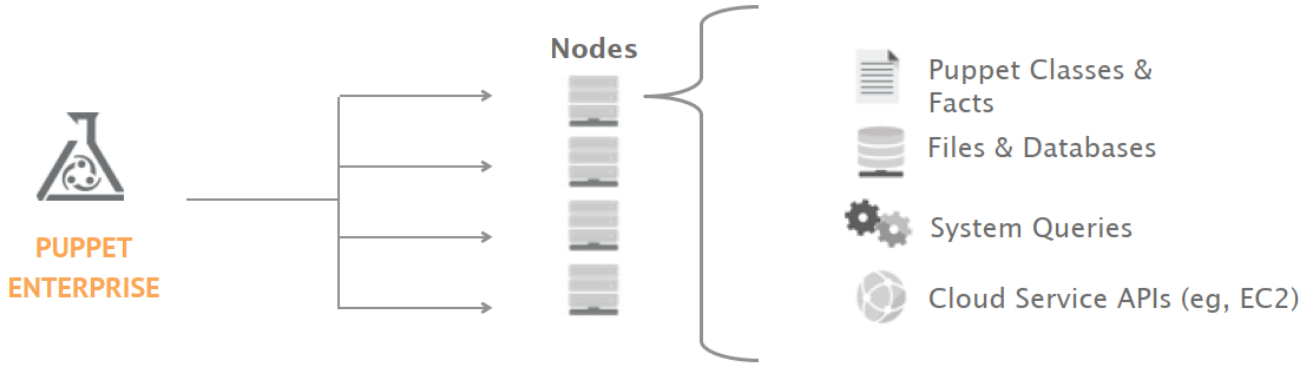
Puppet Enterprise: Discovery

Dynamic, real-time discovery of nodes, resources, and state

Address all nodes **simultaneously**

Query **any data source** on a node

```
% mco find -S "environment=QA and !dept=sales"
```



No More Outdated CMDBs

Current deployment = source of truth

Ask Specific Questions

Focus queries using booleans and regular expressions

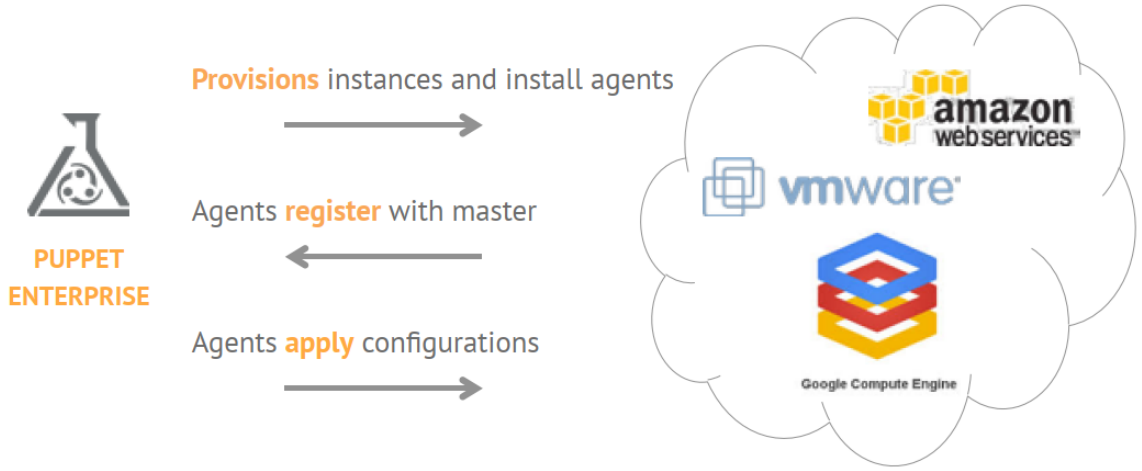
Scalable, Real-time Responses

Asynchronous message bus-based architecture

Puppet Approach

Puppet Enterprise: Provisioning

Quickly stand-up private and public cloud infrastructure



Leverage Existing Work

Re-use on-premise configs for cloud deployments

Many Clouds, One Solution

Avoid lock-in to cloud vendor-specific APIs

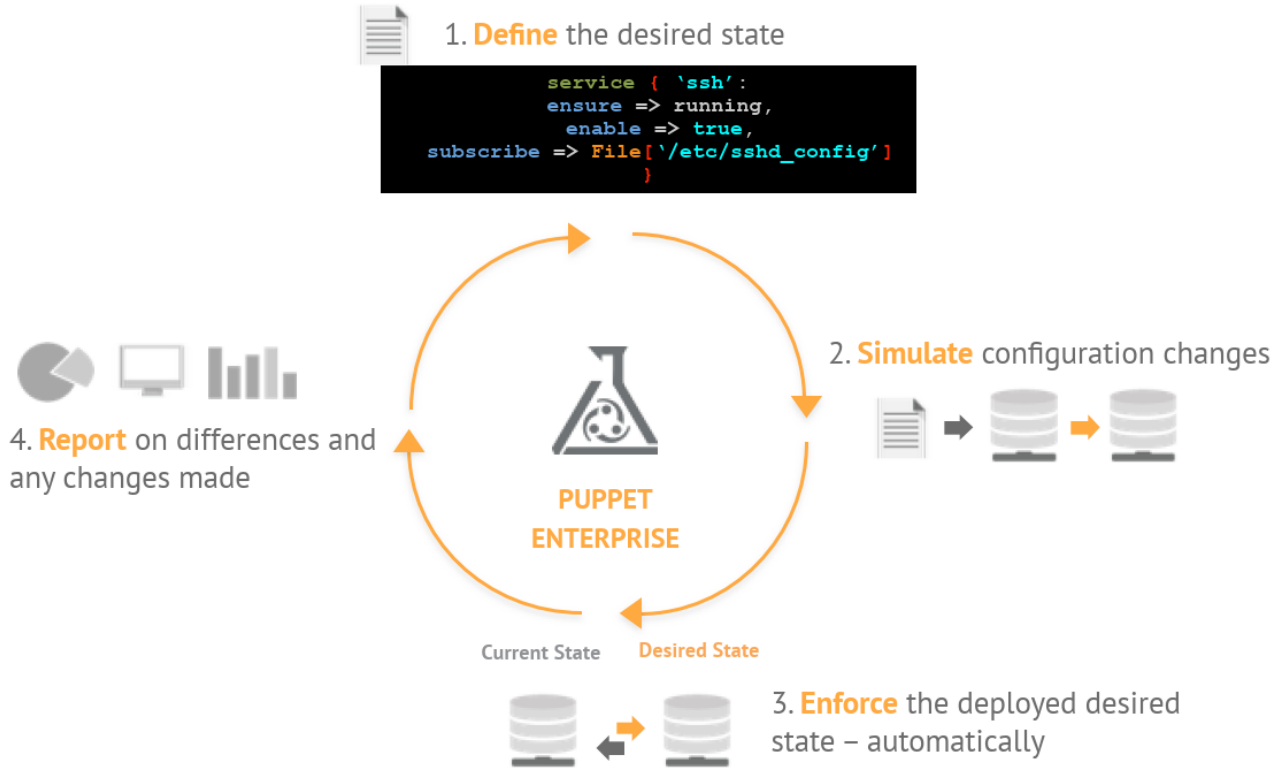
Prevent Cloud Drift

Maintain consistent environments between on-premise and the cloud

Puppet Approach

Puppet Enterprise: Configuration

Improve agility and productivity through defining and enforcing a desired state

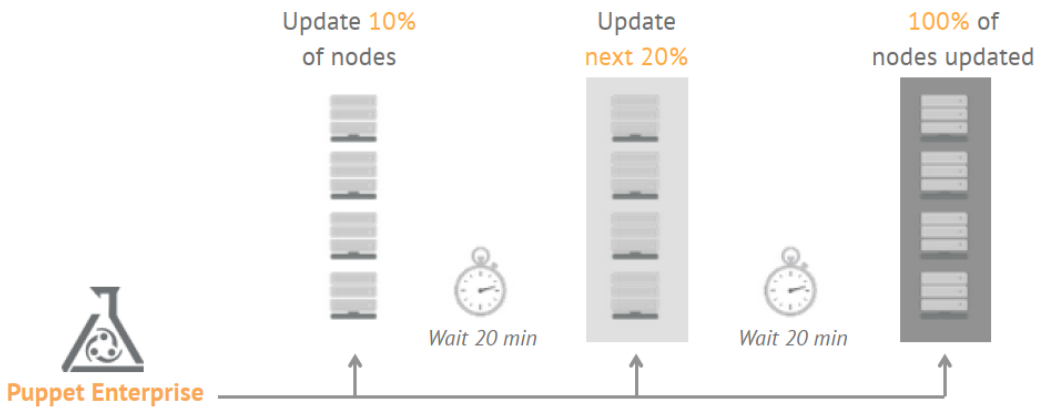


Puppet Approach

Puppet Enterprise: Orchestration

Controlled, multi-step operations to targeted collections of nodes

Goal: update Apache on all QA nodes



Dynamic Multi-step Operations

Chain the outputs of one operation into the next

Manage Change Rate

Progressively apply changes to sub-sets of nodes

Control Change Scope

Apply changes only to specifically tagged nodes

Puppet Approach

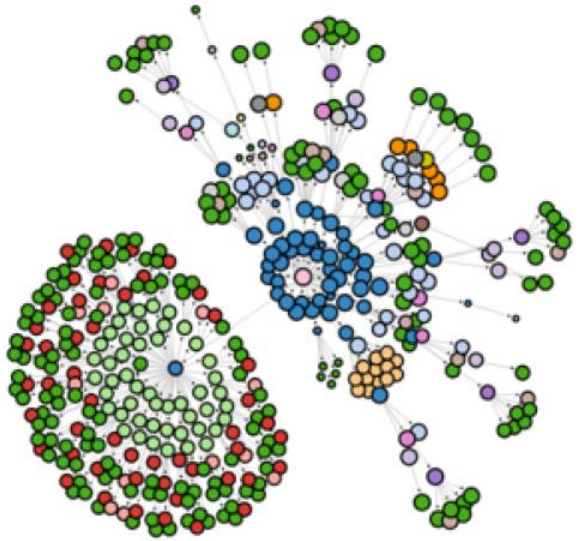
Puppet Enterprise: Reporting

Inventory and change data accessible via GUI and APIs

Comprehensive Infrastructure Data
Hardware and software inventory, change reports, configuration graphs

Open Standards
YAML, JSON, and .dot-formatted data accessible via RESTful API

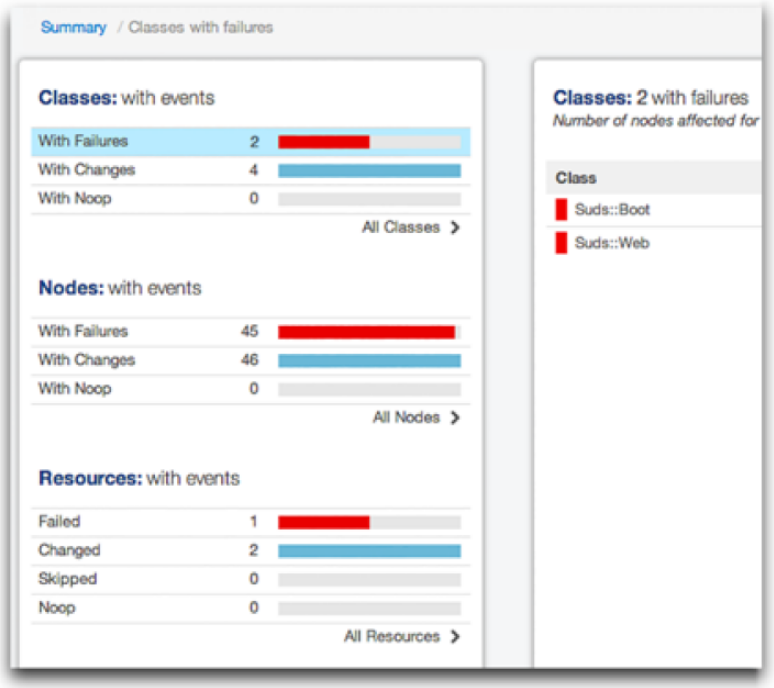
Rich Ecosystem of Tools
Boundary, New Relic, Graphite, GraphViz, Gephi, and many more



Puppet Approach

Puppet Enterprise: Event Inspector

Quickly understand and act on changes occurring in your infrastructure



Know What Changed, Where, & How
Visualize infrastructure changes by Nod Classes, and Resources

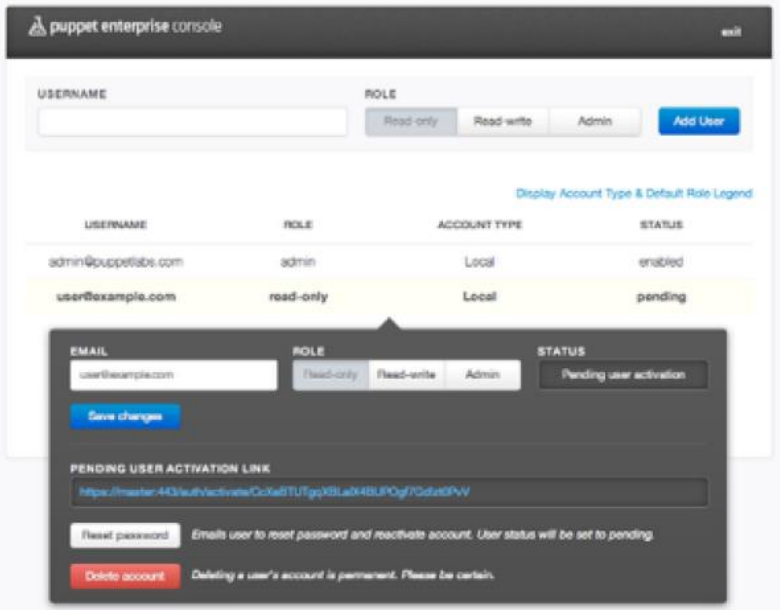
Understand the Impact
Drill-down, zoom-out to evaluate the scope of changes

Take **Action** & Improve Service Levels
Get the specifics to address and manage change

Puppet Approach

Puppet Enterprise: Role-Based Access Control

Read-only, Read-write, and Admin roles



Easy Set-up

Quickly create new users through the Puppet Enterprise console GUI

Easy Installation

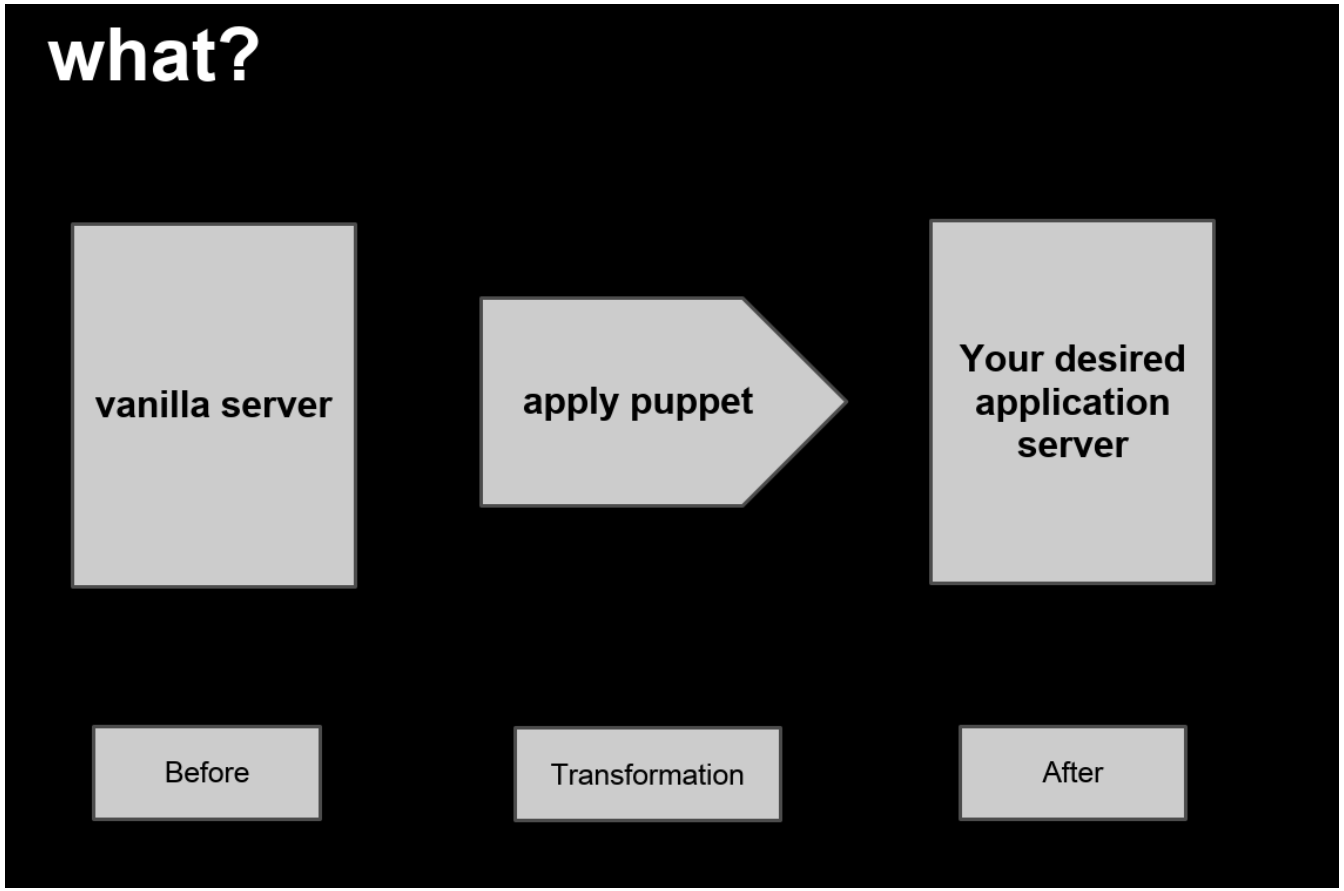
Select from Read-only, Read-write, or Admin roles

Native Resource Support

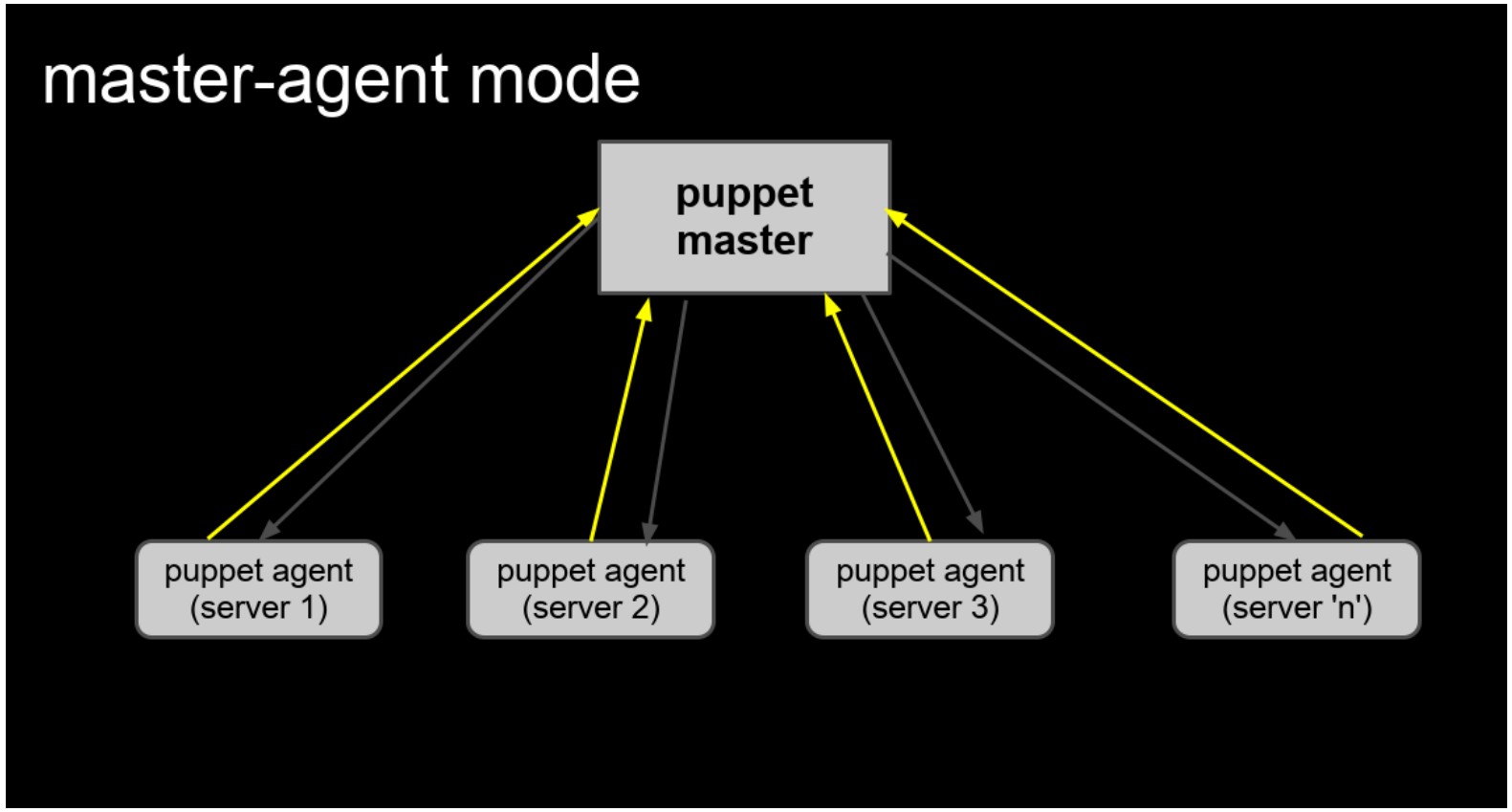
Users' activities logged and auditable

Third-Party Authentication Support LDAP, Active Directory, Google Apps

Puppet Approach



Puppet Approach



Puppet Approach

components

facter & facts
puppet language
resources
manifests
classes
templates

Puppet Approach

facter & facts

puppet uses facter to gather information about the host system

Puppet Approach

puppet language

- DSL
- ruby

Puppet Approach

resources

- the building blocks
- model system configurations
- built-in resources

```
user { 'dave':
   ensure => 'present',
   home => '/home/dave',
   shell => '/bin/zsh'
}
```

- puppet describe -s user

Puppet Approach

resources

- the building blocks
- model system configurations
- built-in resources

```
user { 'dave':
   ensure => 'present',
   home => '/home/dave',
   shell => '/bin/zsh'
}
```

- puppet describe -s user

Puppet Approach

classes

- describe one part of what makes up a system's identity
- not object-oriented programming 'class'

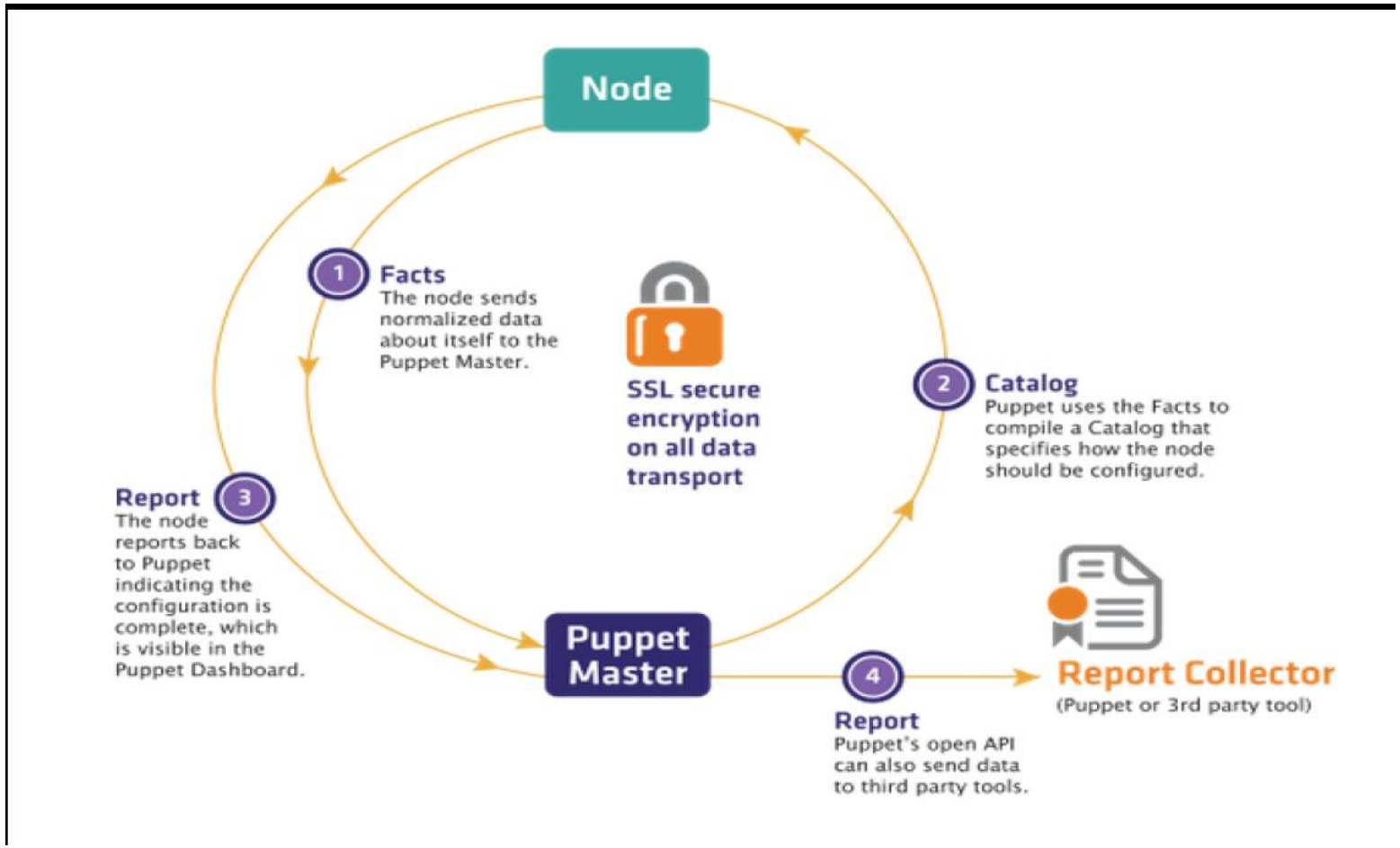
Puppet Approach

templates

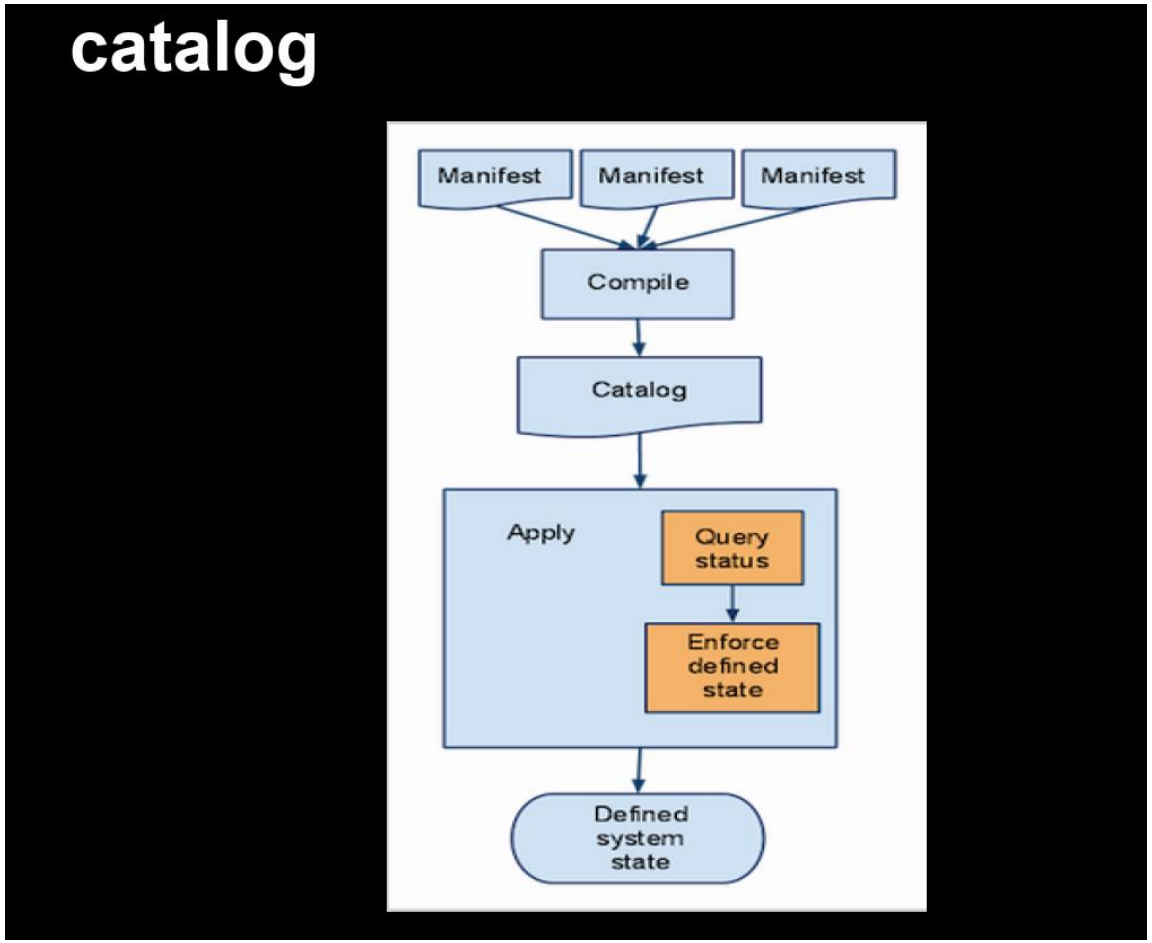
puppet uses ruby erb templating system

```
file {'/etc/foo.conf':
   ensure  => file,
   require => Package['foo'],
   content => template('foo/foo.conf.erb'),
 }
```

Puppet Approach



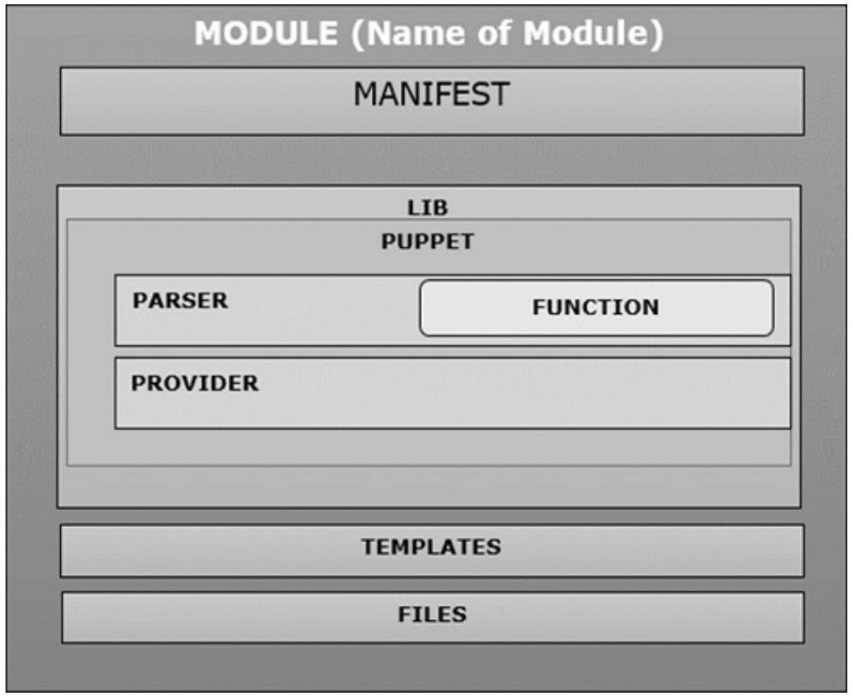
Puppet Approach



Puppet Components

Puppet − Key Components

Following are the key components of Puppet.



Puppet Resources

Puppet resources are the key components for modeling any particular machine. These resources have their own implementation model. Puppet uses the same model to get any particular resource in the desired state.

Providers

Providers are basically fulfillers of any particular resource used in Puppet. For example, the package type ‘apt-get’ and ‘yum’ both are valid for package management. Sometimes, more than one provider would be available on a particular platform. Though each platform always have a default provider.

Manifest

Manifest is a collection of resources which are coupled inside the function or classes to configure any target system. They contain a set of Ruby code in order to configure a system.

Puppet Components

Modules

Module is the key building block of Puppet, which can be defined as a collection of resources, files, templates, etc. They can be easily distributed among different kinds of OS being defined that they are of the same flavor. As they can be easily distributed, one module can be used multiple times with the same configuration.

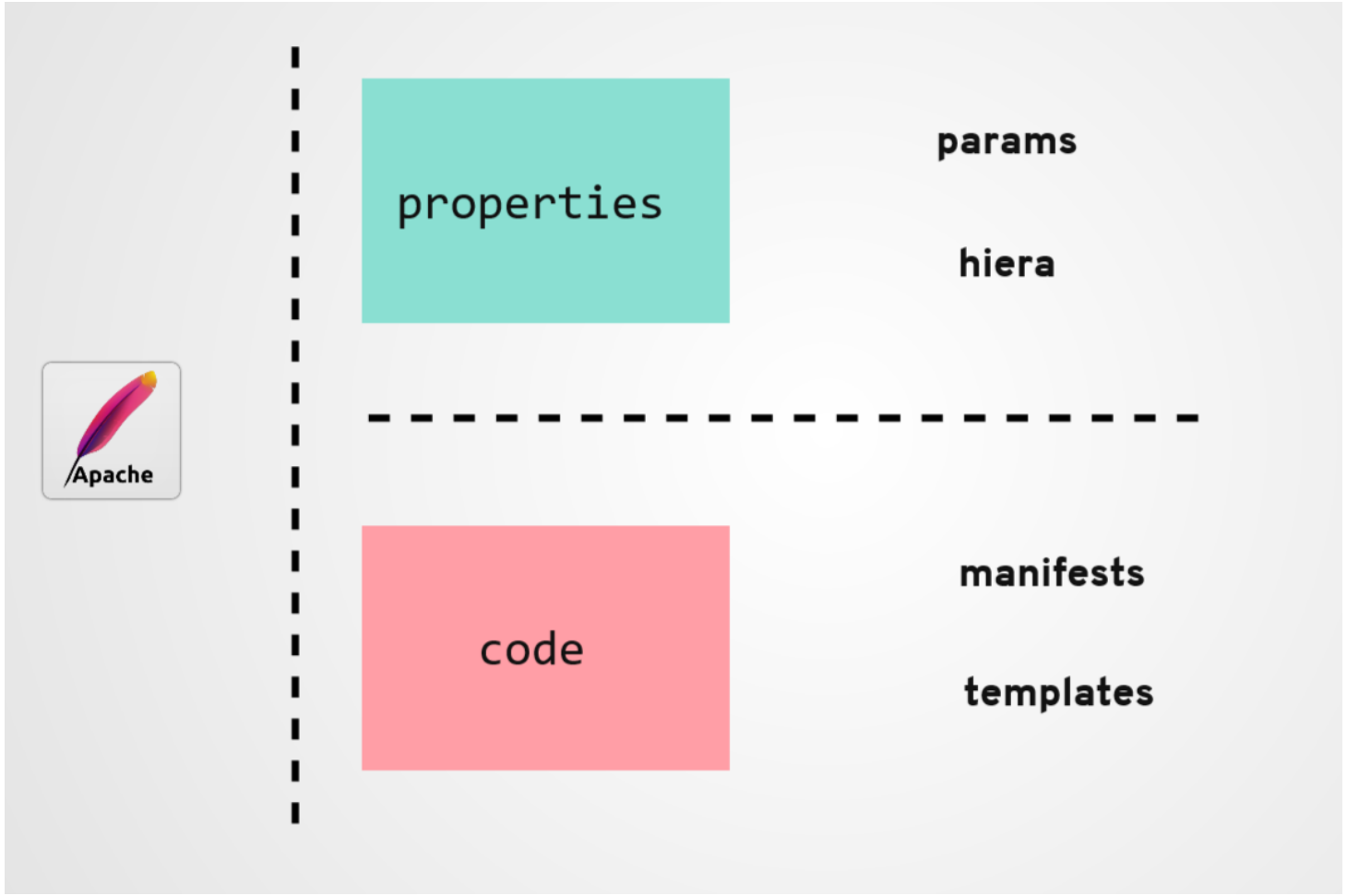
Templates

Templates use Ruby expressions to define the customized content and variable input. They are used to develop custom content. Templates are defined in manifests and are copied to a location on the system. For example, if one wants to define httpd with a customizable port, then it can be done using the following expression.

```
Listen <% = @httpd_port %>
```

Static Files

Static files can be defined as a general file which are sometimes required to perform specific tasks. They can be simply copied from one location to another using Puppet. All static files are located inside the files directory of any module. Any manipulation of the file in a manifest is done using the file resource.


properties
params
hiera
Apache
code
manifests
templates

puppet

Overview of Puppet's architecture

Puppet (PE and open source)

Version 5.3

(↓ expand all ↓) | (↑ close menu ↑)

Puppet Platform reference manual

You can configure systems with Puppet either in a client/server architecture, using the **Puppet agent** and **Puppet master** applications, or in a stand-alone architecture, using the **Puppet apply** application.

Catalogs

A catalog is a document that describes the desired system state for one specific computer. It lists all of the resources that need to be managed, as well as any dependencies between those resources.

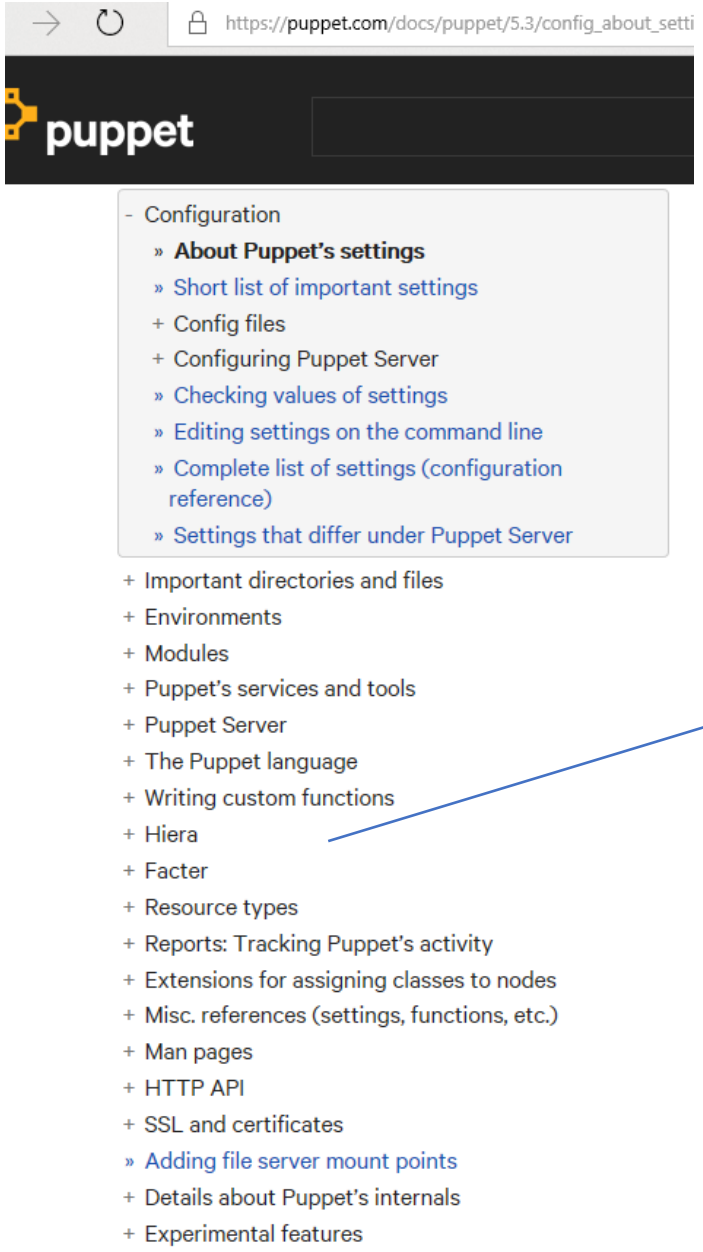
Puppet configures systems in two stages:

1. Compile a catalog.
2. Apply the catalog.

To compile a catalog, Puppet uses several sources of information. For more info, see the pages on basics of the Puppet language and catalog compilation.

The agent/master architecture

When set up as an agent/master architecture, a Puppet master server controls the configuration information, and each managed agent node requests its own configuration catalog from the master.


https://puppet.com/docs/puppet/5.3/config_about_setti
puppet
- Configuration
» About Puppet's settings
» Short list of important settings
+ Config files
+ Configuring Puppet Server
» Checking values of settings
» Editing settings on the command line
» Complete list of settings (configuration reference)
» Settings that differ under Puppet Server
+ Important directories and files
+ Environments
+ Modules
+ Puppet's services and tools
+ Puppet Server
+ The Puppet language
+ Writing custom functions
+ Hiera
+ Facter
+ Resource types
+ Reports: Tracking Puppet's activity
+ Extensions for assigning classes to nodes
+ Misc. references (settings, functions, etc.)
+ Man pages
+ HTTP API
+ SSL and certificates
» Adding file server mount points
+ Details about Puppet's internals
+ Experimental features
- Hiera
» Hiera
» Getting started with Hiera
» Configuring Hiera
» Creating and editing data
» Looking up data with Hiera
» Writing new data backends
» Upgrading to Hiera 5

Hiera is a key/value lookup used for separating data from Puppet code.

About Hiera

Hiera is Puppet's built-in key-value configuration data lookup system.

Puppet's strength is in reusable code. Code that serves many needs must be configurable: put site-specific information in external configuration data files, rather than in the code itself.

Puppet uses Hiera to do two things:

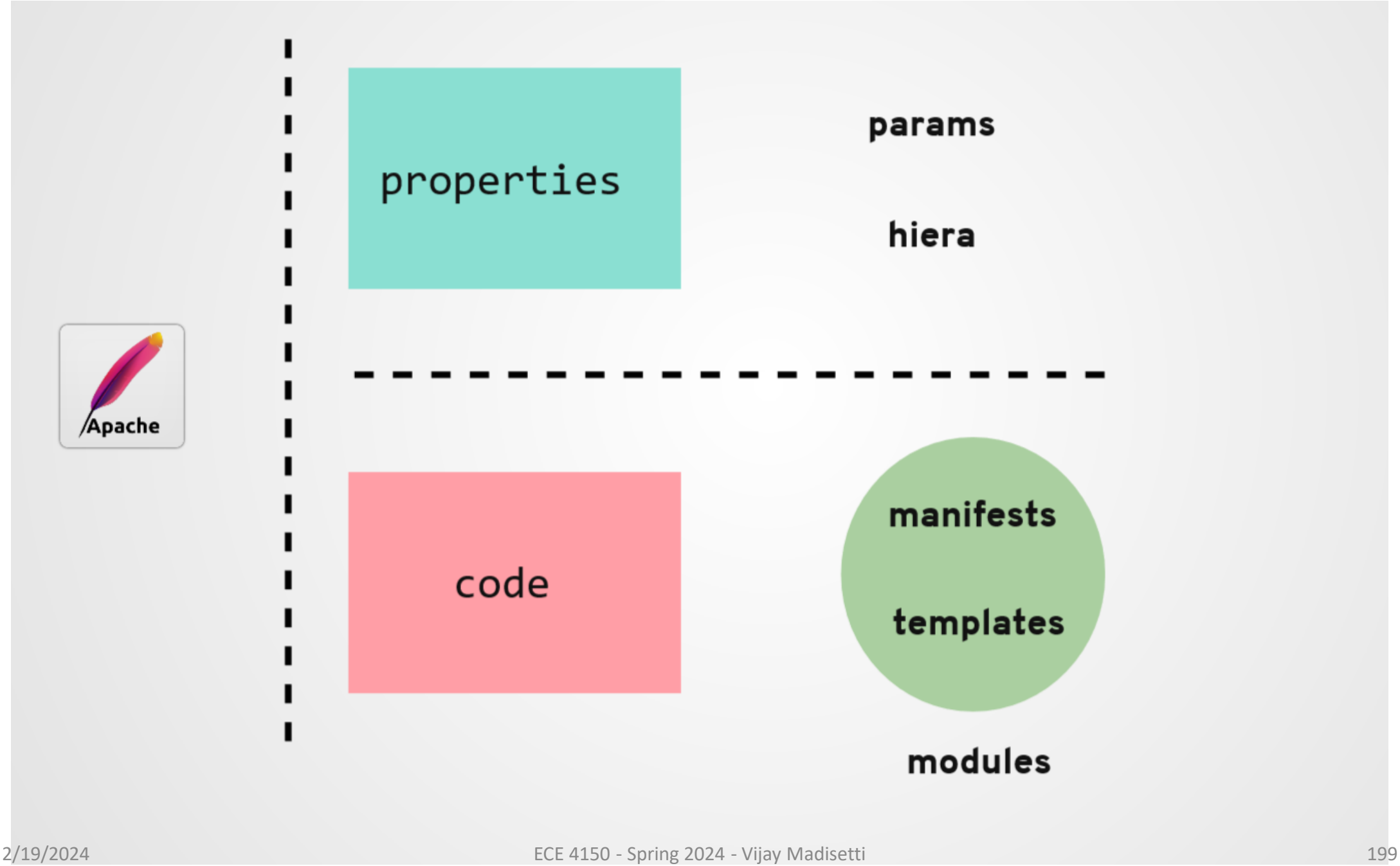
- Store the configuration data in key-value pairs
- Look up what data a particular module needs for a given node during catalog compilation.

This is done via:

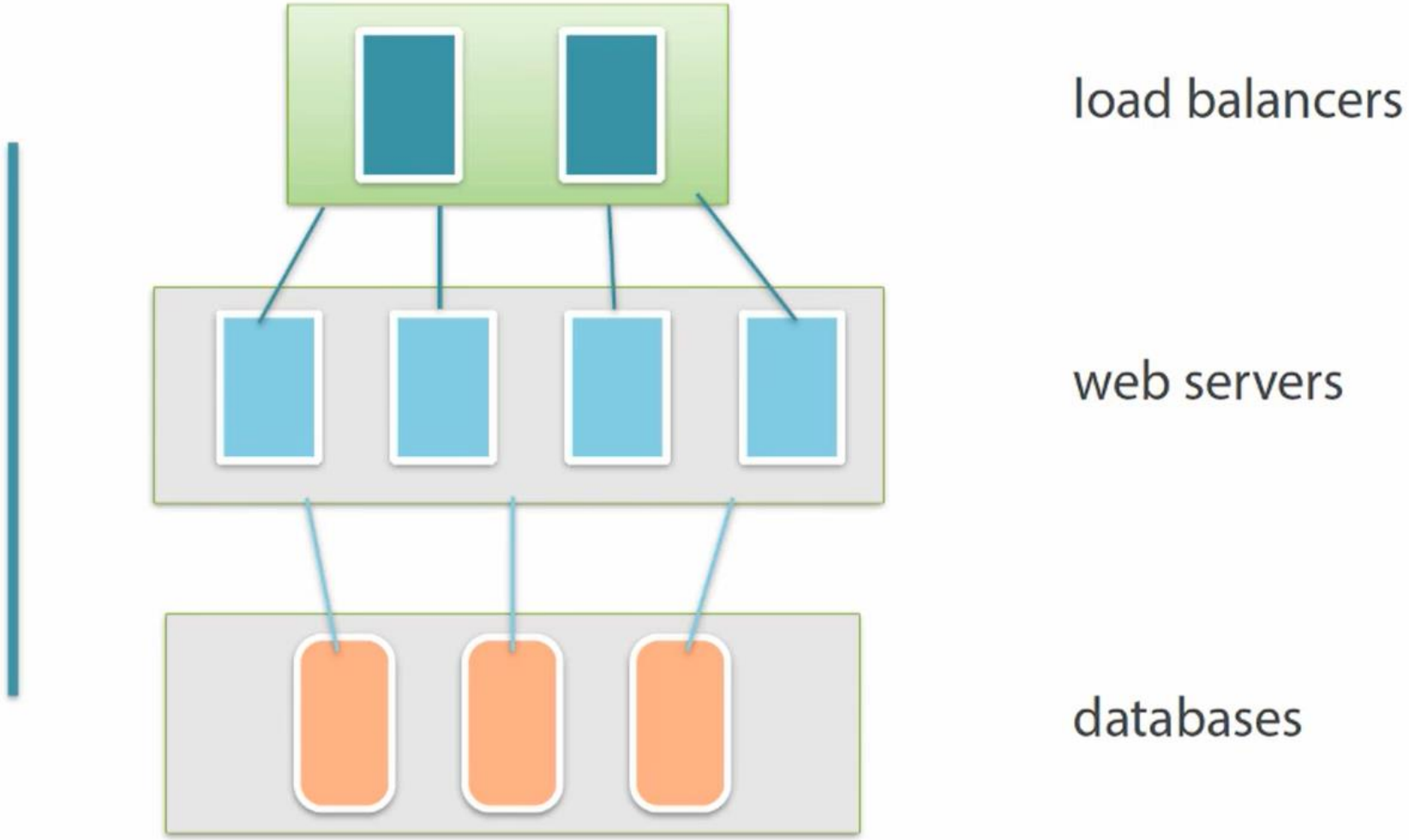
- Automatic Parameter Lookup for classes included in the catalog
- Explicit lookup calls

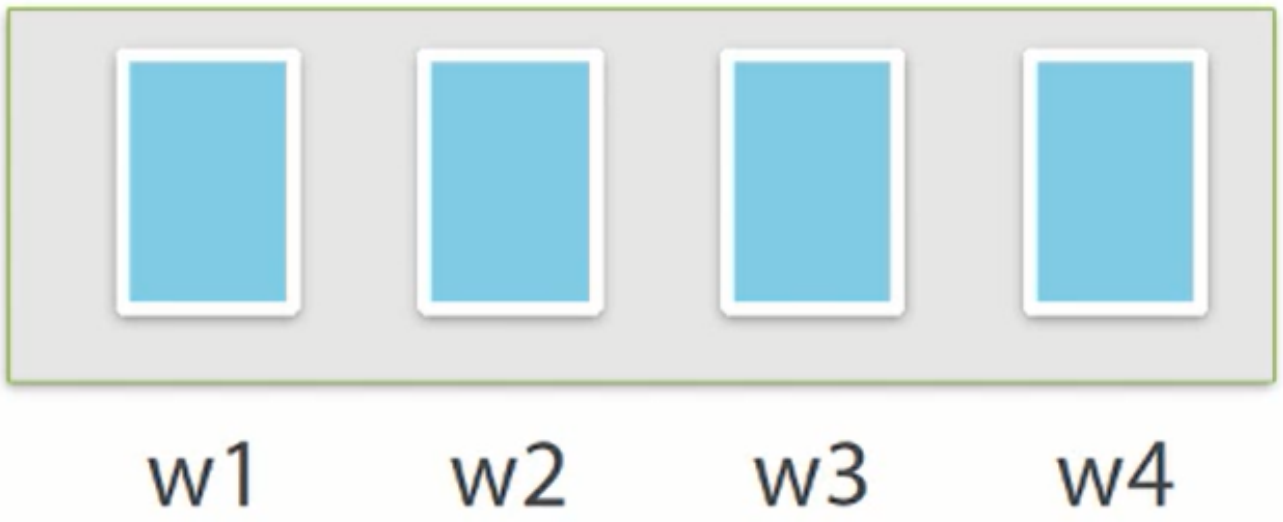
Hiera's hierarchical lookups follow a "defaults, with overrides" pattern, meaning you specify common data once, and override it in situations where the default won't work. Hiera uses Puppet's facts to specify data sources, so you can structure your overrides to suit your infrastructure. While using facts for this purpose is common, data-sources may well be defined without the use of facts.

Hiera 5 comes with support for JSON, YAML, and EYAML files.


properties
params
hiera
Apache
code
manifests
templates
modules

End State





web servers

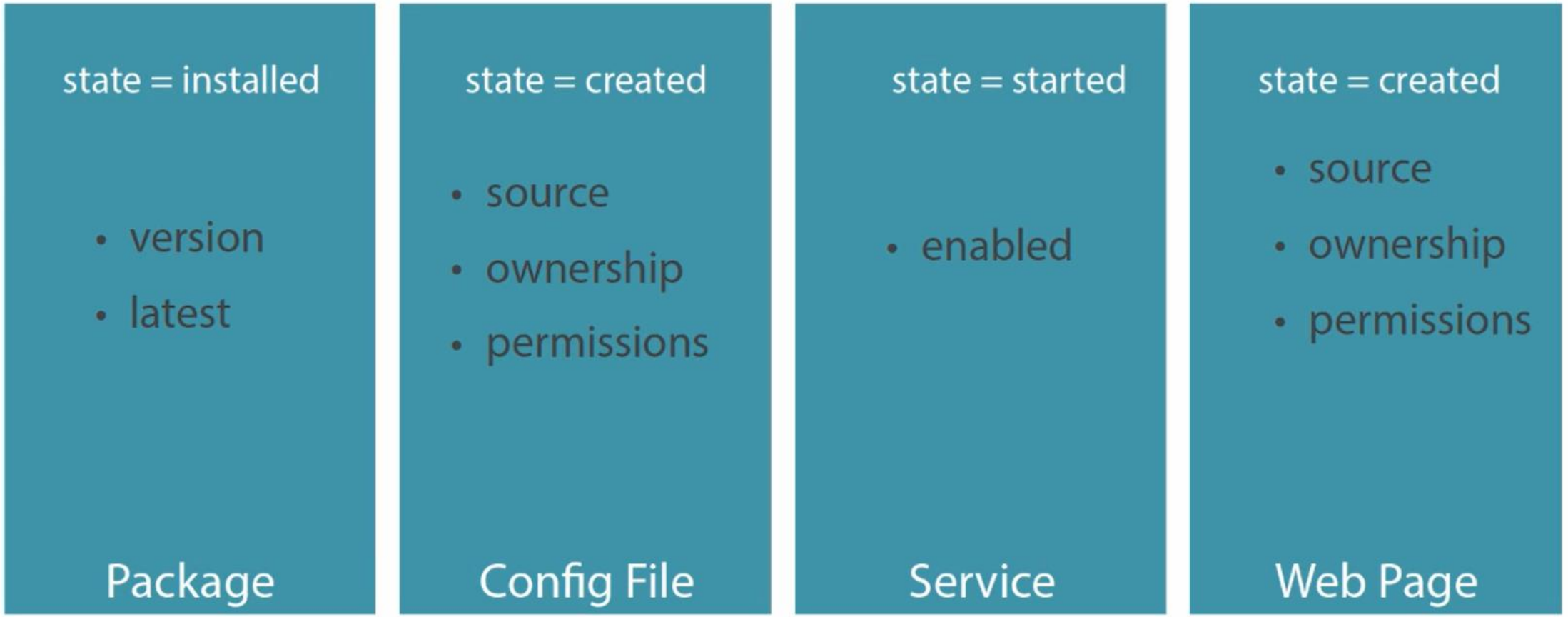
nginx

- Install **Package**
- Create Configuration **File**
- Start **Service**
- Create **Web Page**

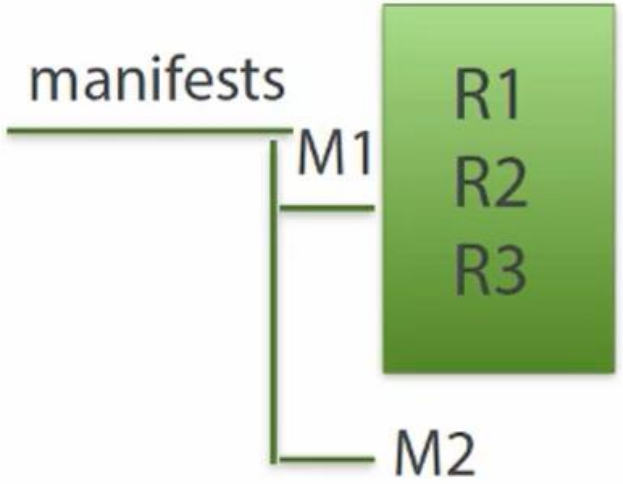
- Install **Package**
- Create Configuration **File**
- Start **Service**
- Create **Web Page**

- **Package**
- **File**
- **Service**
- **File**

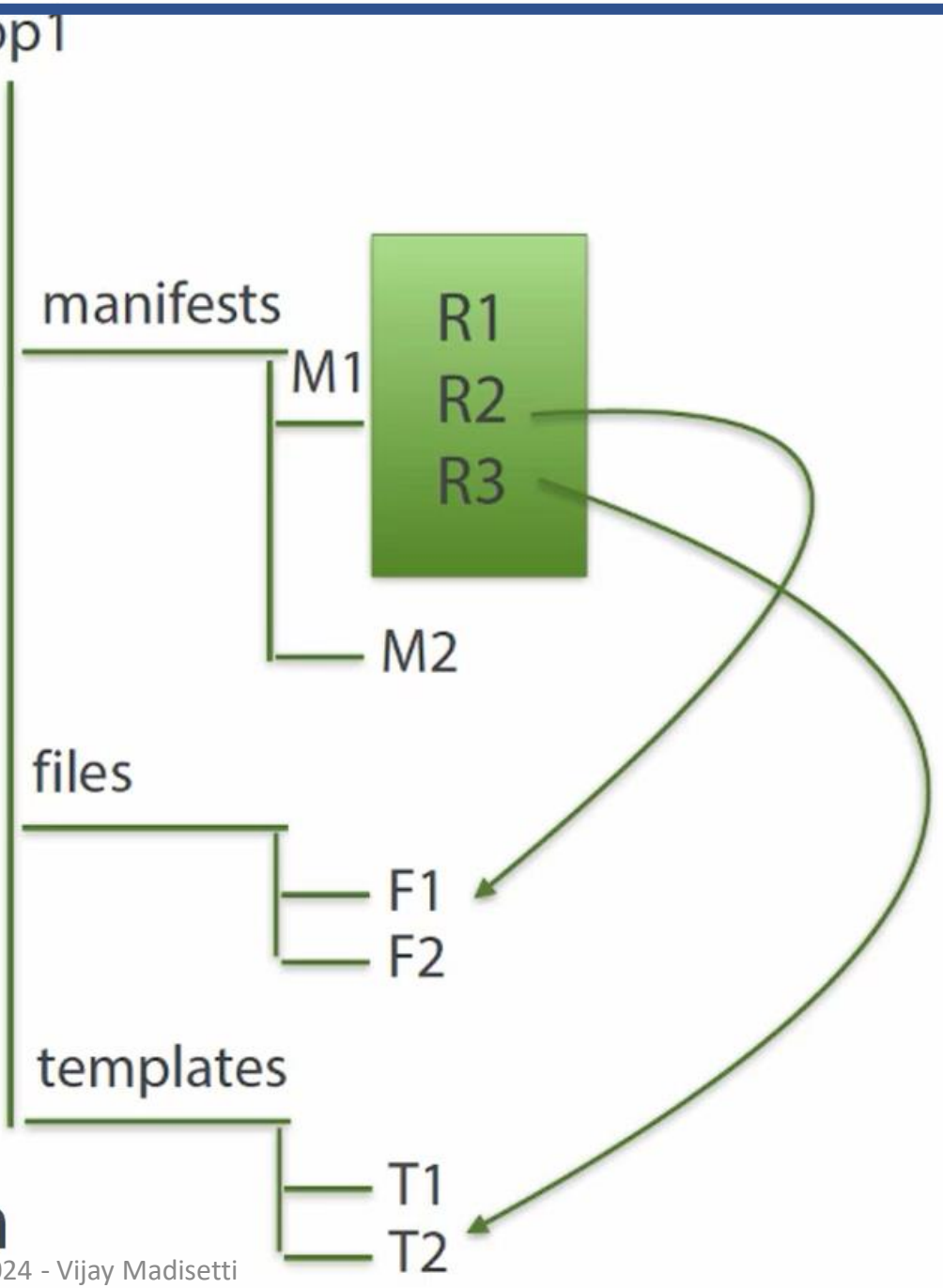
Nginx Blueprint with Puppet



Manifests Directory



Module

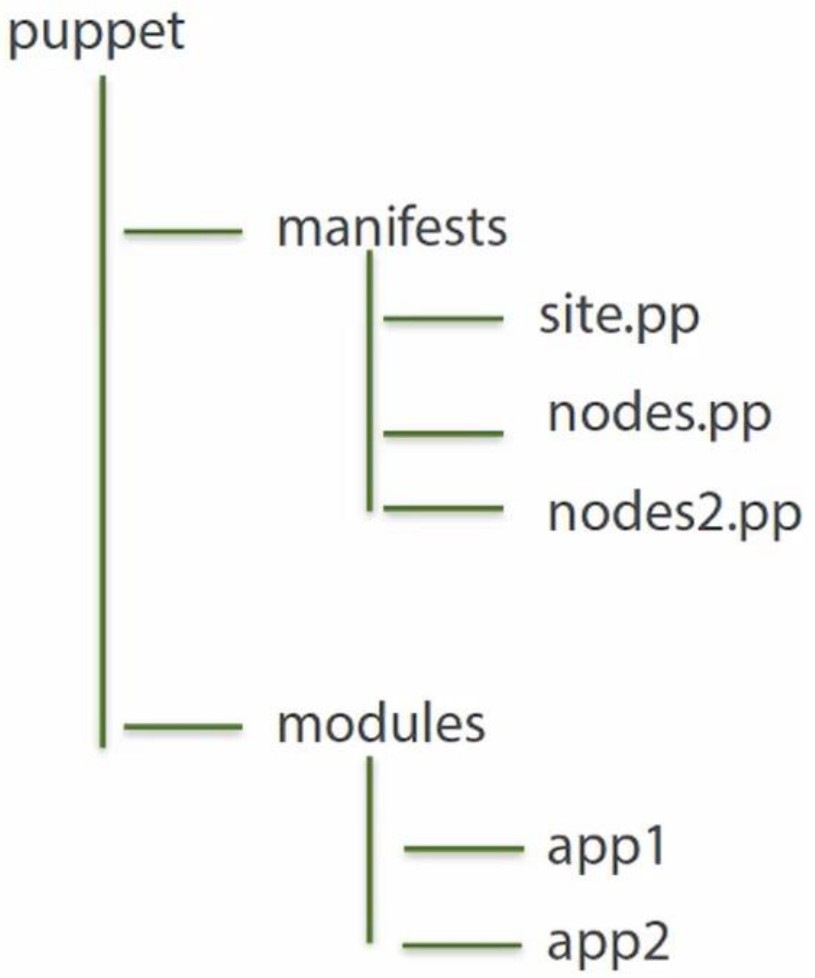


Modules are units of Automation

nodes.pp

```
node 'demo' {
    include app1
}
```

Modules Directory



RESOURCES

these entities can then be described using **resources.**

resources are statements of configuration policy

which are written using

PUPPET'S DSL

(Domain specific language)

RESOURCES

Puppet then translates these **resources** into **providers** which are platform specific procedures

package

yum

apt

zypper

https://docs.puppet.com/puppet/latest/type.html

Resource Type Reference (Single-Page)

Puppet (PE and open source)

Version 5.3

(↓ expand all ↓) | (↑ close menu ↑)

Puppet Platform reference manual

Attributes

```
computer { 'resource title':
  name        => # (namevar) The authoritative 'short' name of the computer...
  ensure      => # Control the existences of this computer record...
  en_address  => # The MAC address of the primary network...
  ip_address  => # The IP Address of the Computer...
  provider    => # The specific backend to use for this `computer...
  realname    => # The 'long' name of the computer...
  # ...plus any applicable metaparameters.
}
```

name

(***Namevar:*** *If omitted, this attribute's value defaults to the resource's title.)*

The authoritative 'short' name of the computer record.

(↑ Back to computer attributes)

ensure

(***Property:*** *This attribute represents concrete state on the target system.)*

Control the existences of this computer record. Set this attribute to `present` to ensure the computer record exists. Set it to `absent` to delete any computer records with this name

Valid values are `present`, `absent`.

(↑ Back to computer attributes)

Description

Manages files, including their content, ownership, and permissions.

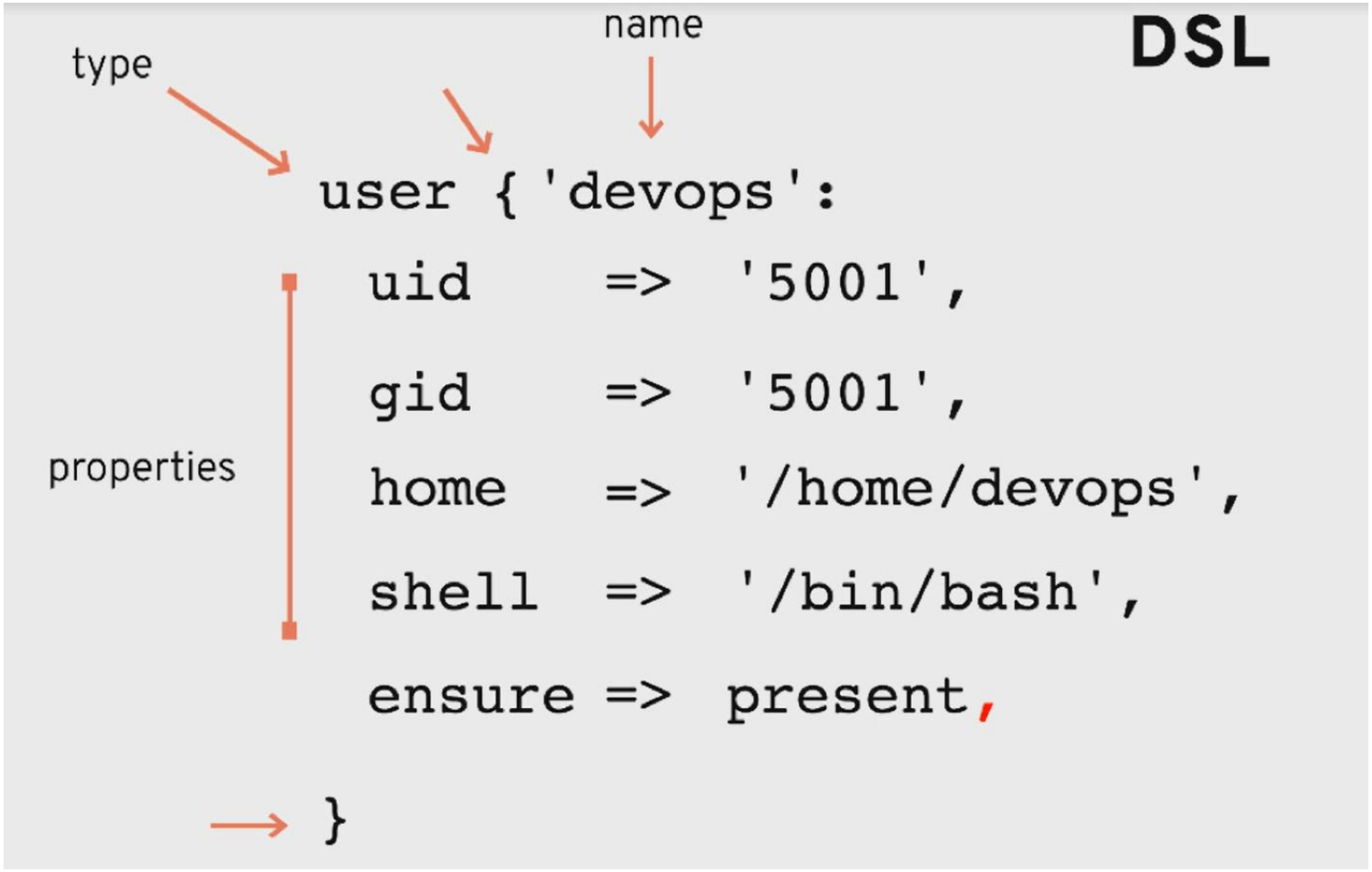
The `file` type can manage normal files, directories, and symlinks; the type should be specified in the `ensure` attribute.

File contents can be managed directly with the `content` attribute, or downloaded from a remote source using the `source` attribute; the latter can also be used to recursively serve directories (when the `recurse` attribute is set to `true` or `local`). On Windows, note that file contents are managed in binary mode; Puppet never automatically translates line endings.

Autorequires: If Puppet is managing the user or group that owns a file, the file resource will autorequire them. If Puppet is managing any parent directories of a file, the file resource will autorequire them.

Attributes

```
file { 'resource title':
  path                    => # (namevar) The path to the file to manage.  Must be fully…
  ensure                  => # Whether the file should exist, and if so what…
  backup                  => # Whether (and how) file content should be backed…
  checksum                => # The checksum type to use when determining…
  checksum_value          => # The checksum of the source contents. Only md5…
  content                 => # The desired contents of a file, as a string…
  ctime                   => # A read-only state to check the file ctime. On…
  force                   => # Perform the file operation even if it will…
  group                   => # Which group should own the file.  Argument can…
  ignore                  => # A parameter which omits action on files matching
  links                   => # How to handle links during file actions.  During
  mode                    => # The desired permissions mode for the file, in…
  mtime                   => # A read-only state to check the file mtime. On…
  owner                   => # The user to whom the file should belong…
  provider                => # The specific backend to use for this `file`…
  purge                   => # Whether unmanaged files should be purged. This…
  recurse                 => # Whether to recursively manage the _contents_ of…
  recurselimit            => # How far Puppet should descend into…
  replace                 => # Whether to replace a file or symlink that…
  selinux_ignore_defaults => # If this is set then Puppet will not ask SELinux…
  selrange                => # What the SELinux range component of the context…
  selrole                 => # What the SELinux role component of the context…
  seltype                 => # What the SELinux type component of the context…
  seluser                 => # What the SELinux user component of the context…
  show_diff               => # Whether to display differences when the file…
  source                  => # A source file, which will be copied into place…
  source_permissions      => # Whether (and how) Puppet should copy owner…
  sourceselect            => # Whether to copy all valid sources, or just the…
  target                  => # The target for creating a link.  Currently…
  type                    => # A read-only state to check the file…
  validate_cmd            => # A command for validating the file's syntax…
  validate_replacement    => # The replacement string in a `validate_cmd` that…
  # ...plus any applicable metaparameters.
}
```


DSL
type
name
user { 'devops':
uid => '5001',
gid => '5001',
home => '/home/devops',
shell => '/bin/bash',
ensure => present,
properties
}

MANIFESTS

```
user {"deploy" :
  ensure     => present,
  uid        => 5001,
  password   => '$1$WD98.uaZ$cx
  home       => '/home/deploy'
}

user {"dojo" :
  ensure   => absent,
}

package { "tree":

  ensure   => installed

}
```

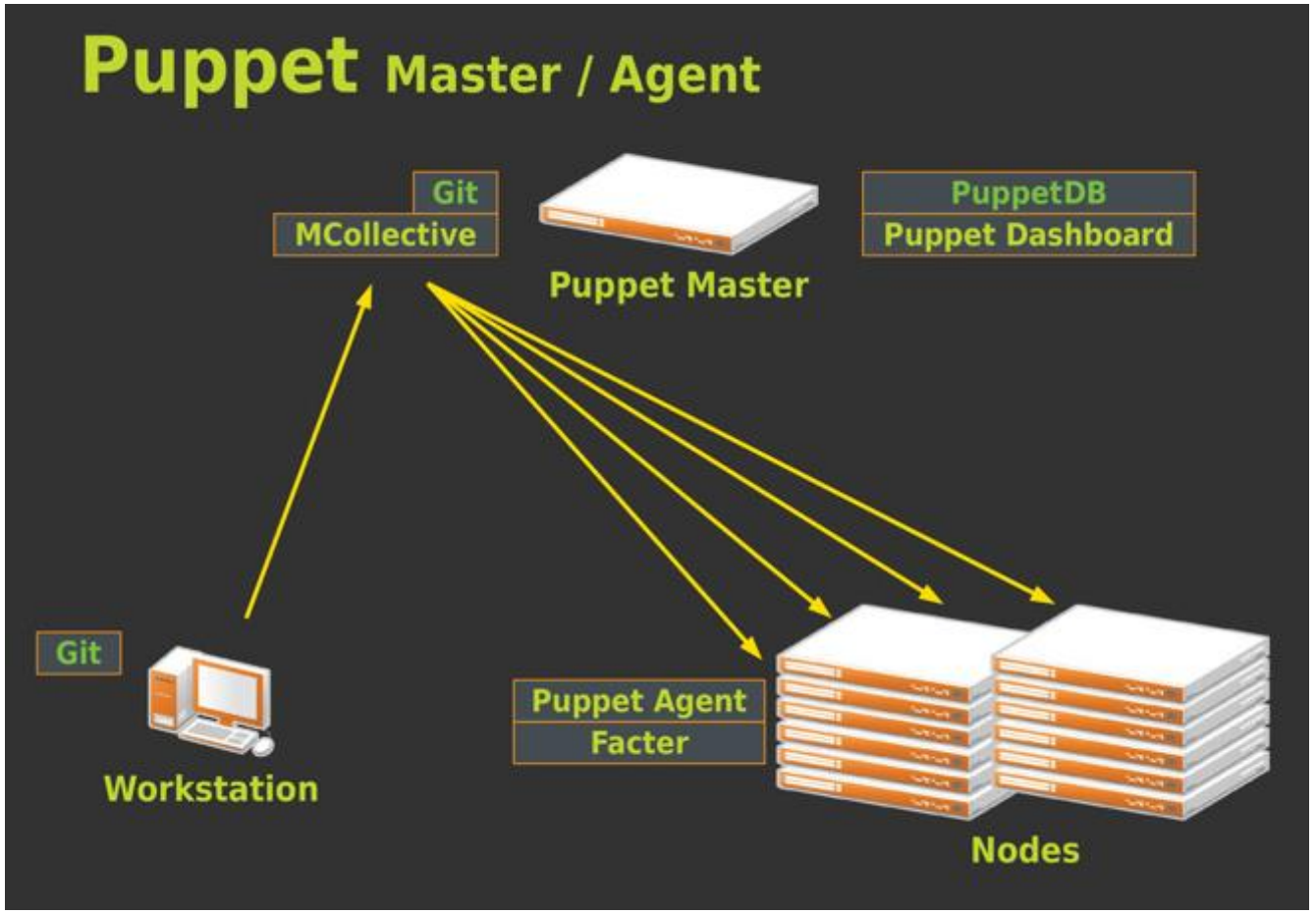
- manifests are files which contain collection of resources
- written to achieve a specific objective
- have .pp extension

Puppet and Chef are Similar in Terms of Architecture of IaaC

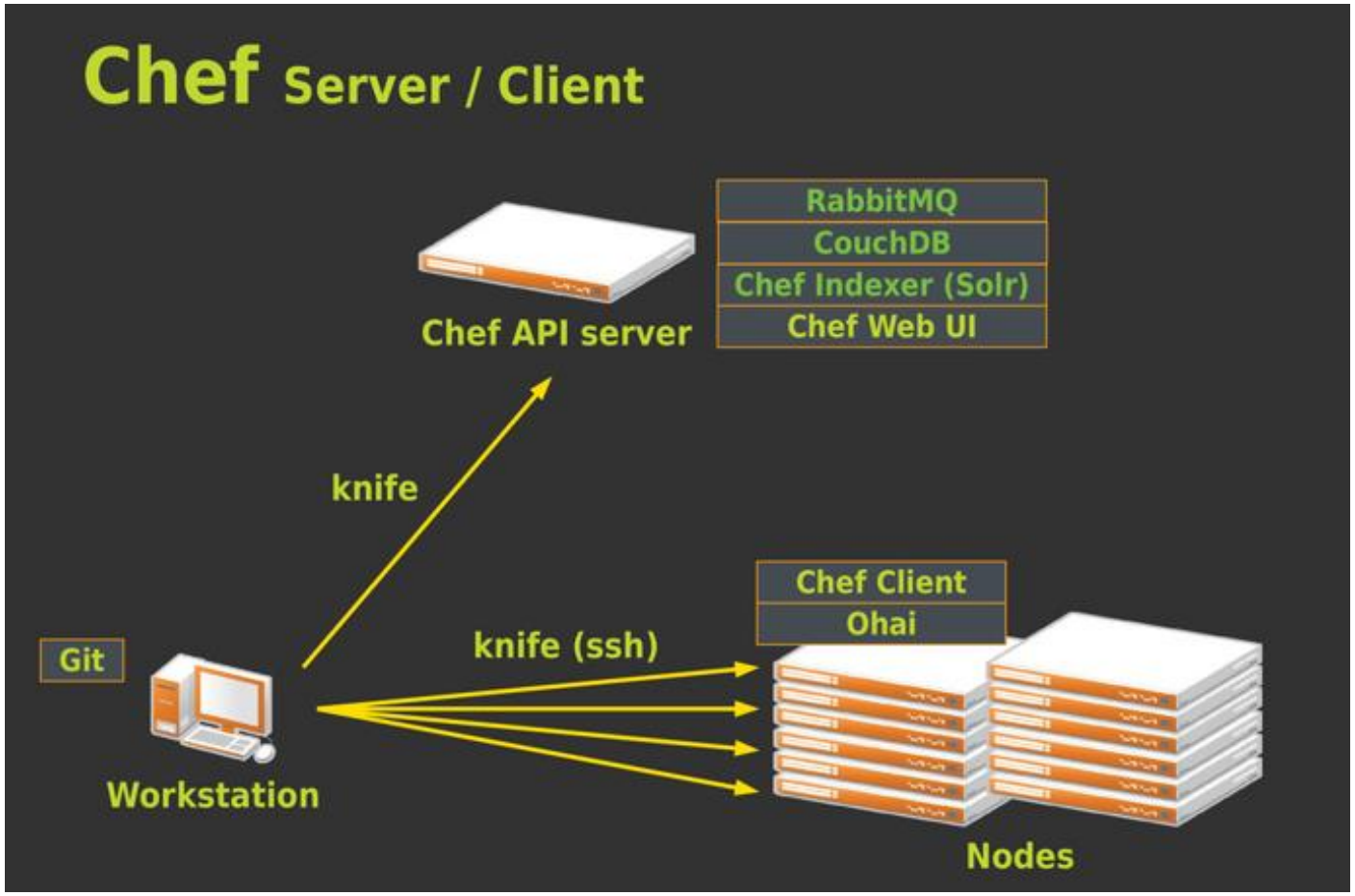
Code reuse

Puppet modules	Chef cookbooks
Manifests	Recipes
Files	File Distribution
Libraries	Libraries
Templates (ERB)	Templates (ERB)
	Attributes
	Definitions
	Lightweight Resources & Providers (LWRP)
	Metadata

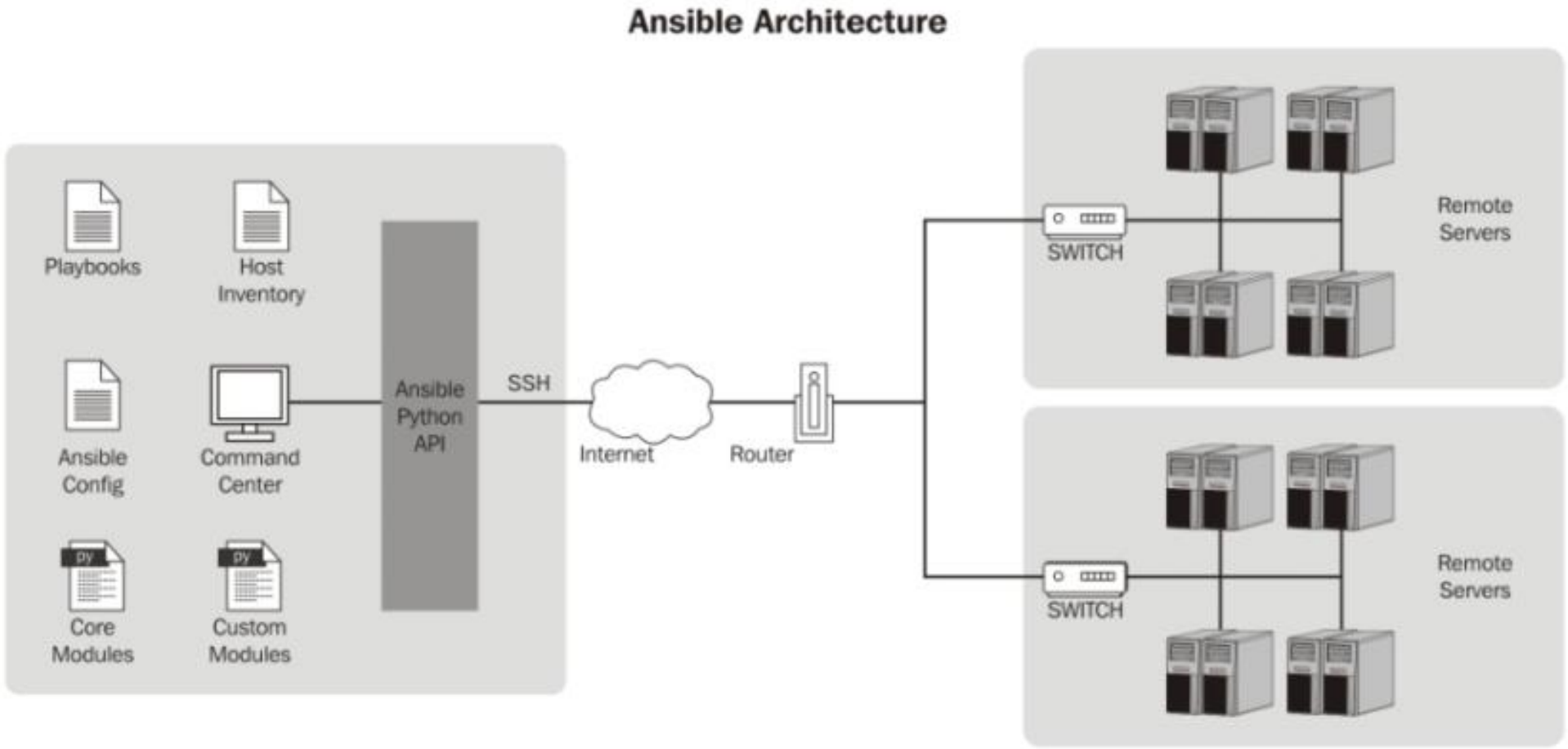
Puppet Architecture



Chef Architecture

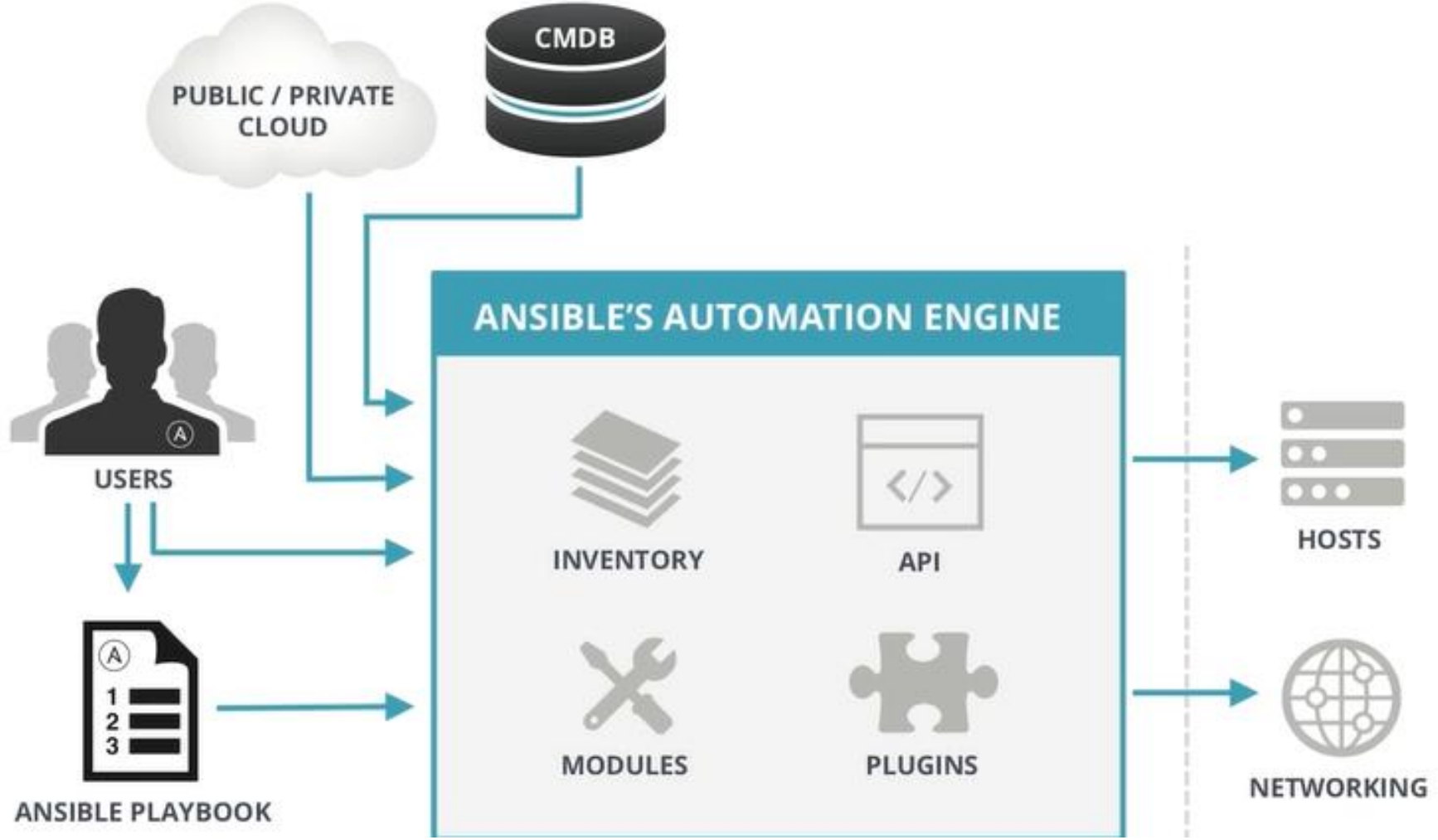


Ansible Architecture

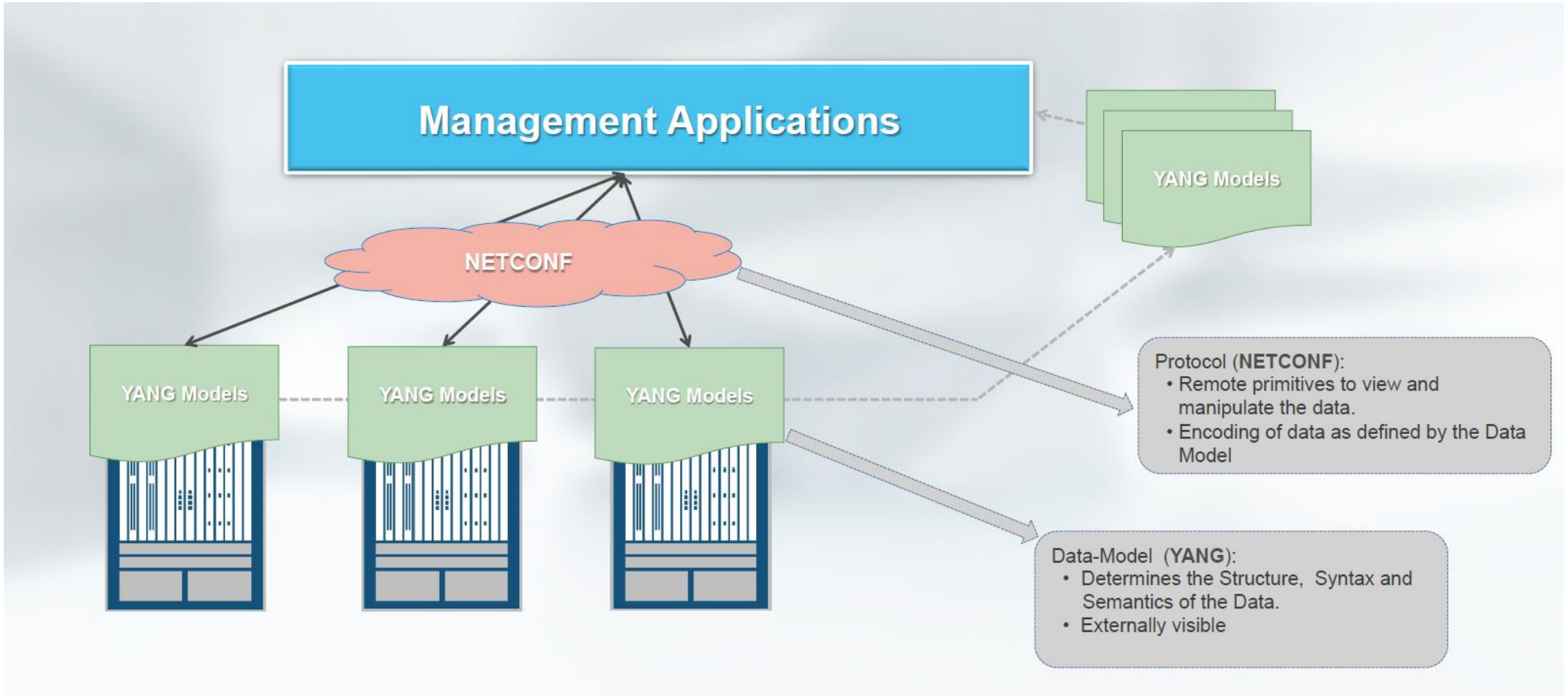


ANSIBLE ARCHITECTURE

ANSIBLE



Another Promising Approach – Netconf and Yang (language)



YANG – Data Model

Data Model Provides Rules for Data

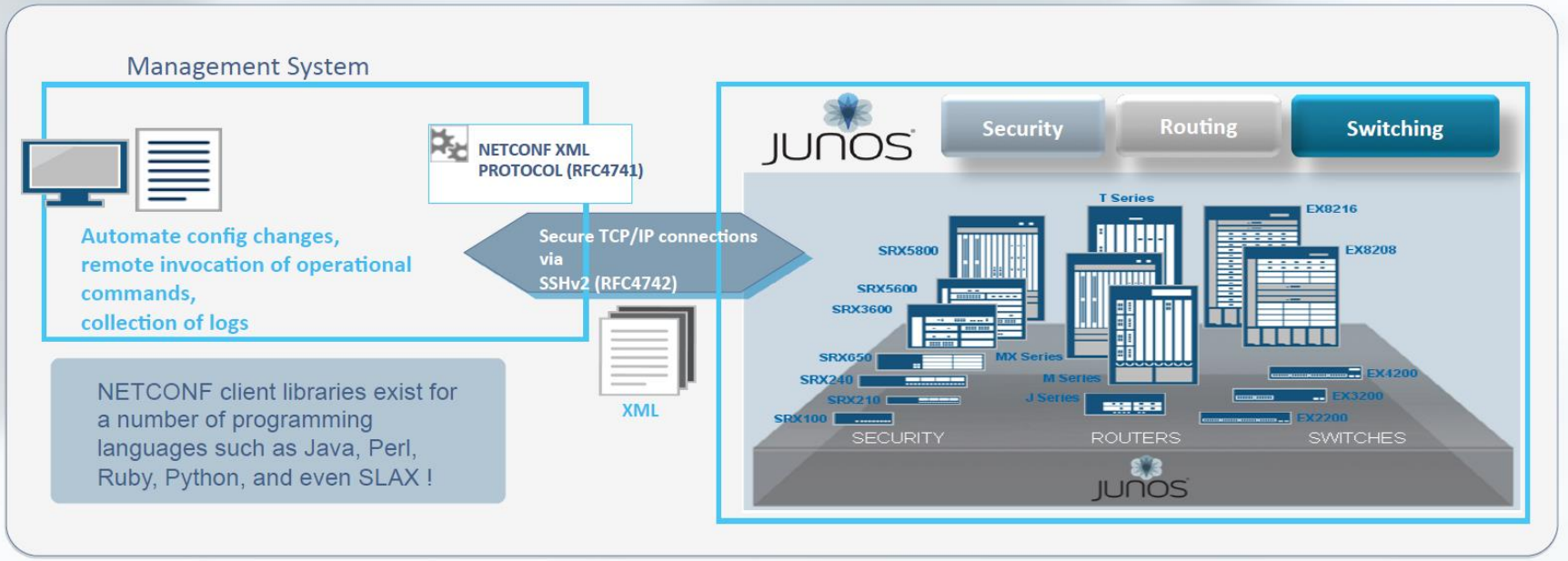
Details all aspects of the data

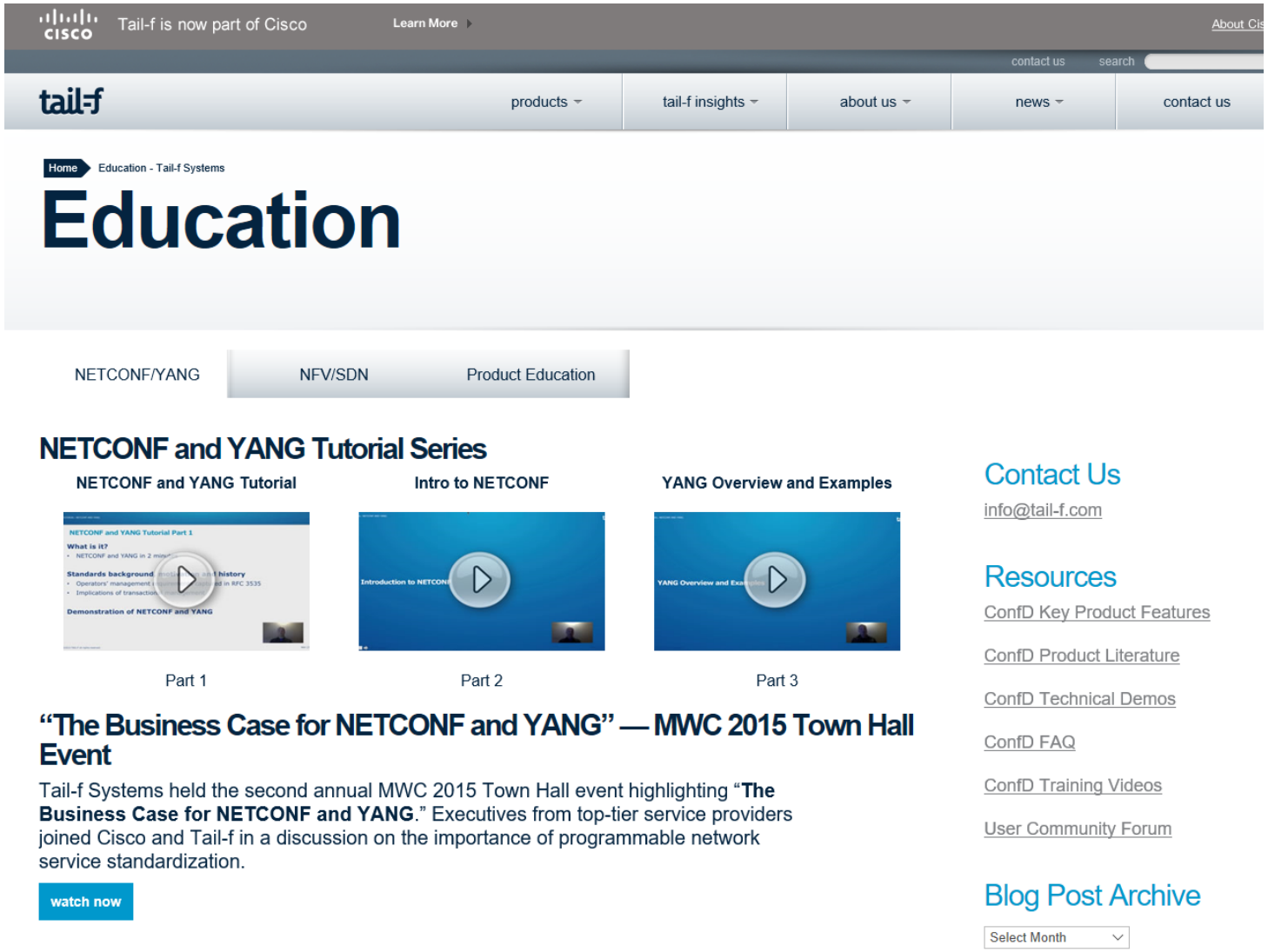
What values are acceptable?	How is the data organized?	How do I manipulate it?	What does the data mean?
• Data types • Ranges • Lengths • Regex matches	• Hierarchy • Identifiers • Uniqueness • Ordering • Referential integrity	• CRUD operations • merge, replace, • insert before/after	• Documentation • References

Software Defined Networks with YANG/NETCONF

Off-box Automation with NETCONF Toolkits

- Secure and connection oriented … SSHv2 as transport
- Structured and transaction based … XML as RPC request / response
- User-class privilege aware … Native to Junos





Cisco's Use of Netconf & YANG

Summary

- IaaC is a powerful concept – software defined systems using declarative approach
- Kubernetes, Chef, Puppet and Ansible are leaders in this area
- YAML, Netconf and YANG are declarative approaches as well
- Allow description of desired state – convergent approach, as opposed to “how to” or low-level scripts that are difficult to manage at scale