Cloud Native Fundamentals

2.1 Introduction to Cloud Native Fundamentals

Cover
1. architecture considerations
2. container orchestration
3. open source PaaS
4. Cloud Native CI/CD

Cloud native
- build at scale
- on-premise, hybrid or public cloud
- speed + agility

Containers
- small unit of applic
- deploy fast, manageable
- fits with microservice-based architecture

Container Orchestration - manage containers
- Docker Swarm
- Apache Mesos
- Kubernetes - automates - ~~configure~~ management scalability
  - runtime
  - network
  - storage
  - servicemesh
  - logs + metrics
  - tracing

2-2 Architecture Considerations

Cover
1. Monolith + microservices
2. Code-db
3. Practices for applicat development

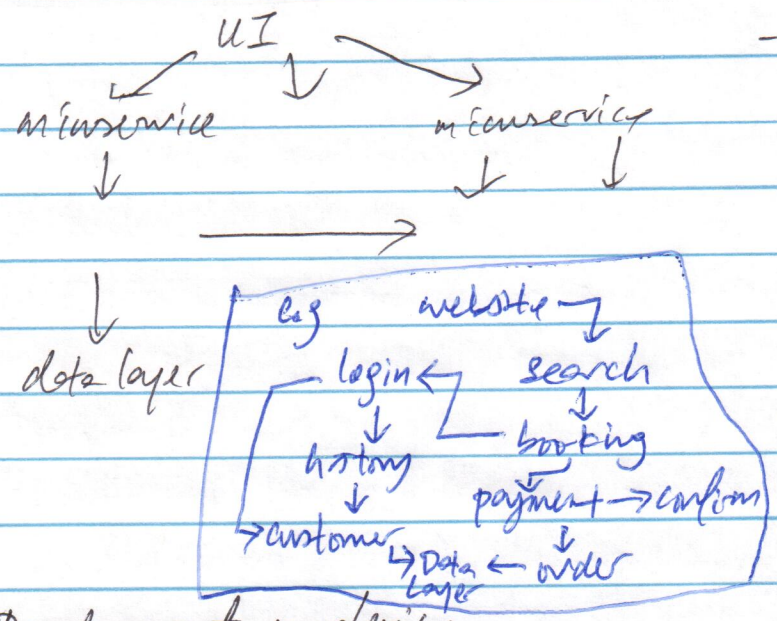
Design
- Requirements - stakeholders
  - functionalities
  - end-users
  - input, output processes
  - engineering teams
- Available resources - engineering resources
  - budget
  - time frames
  - internal knowledge

Applicat architecture
- monolith, microservice
- applicat tiers - UI - HTTP requests + response
  - business logic - services to users
  - data layer - access + storage of data layers
- ~~Mono~~ monolith
  - UI
  - Business
  - Data layer
  - same arch
  - same repository, shared resources, same programming language → single binary

smaller units

, application design where applications are independent

- Microservices
  - small, independent units
  - separate services
    - allocated resources
    - each exposes API
    - each programming language of choice
    - concurrent development
    - separate repository, separate binaries



Development complexity

Monolith	Microservices
- 1 language	- >1 languages
- 1 repository	- >1 repositories
- sequential	- concurrent
Scalability	
- replicate of entire stack	- replicate of individual functionality
- overconsumption of resources	- on-demand resources consumption
Time to deploy	
- 1 delivery pipeline	- multiple delivery pipelines
- entire stack development	- separate function deployment
failure on 1, brings down entire application	
	- high velocity at scale

Flexibility	
- low	- high
Operational Cost	
- low initial cost	- high initial cost
- high cost at scale	- low cost at scale
Reliability	
- recovery of entire stack	- recovery of failed component only
- low visibility into logs	- high visibility into logs

Best practices

- Health checks - status report
  - /health or /status
  - return response - 'healthy' or 'error'
- Metrics - statistics for individual services (CPU, memory, network throughput
  - used + handled requests
  - /metrics
- Logs - record operations
  - return response
  - passive logging: STDOUT, STDERR
  - active logging: Direct interface
- Tracing - records requested lifecycle
  - application layer implementation

DEBUG, INFO, WARN, ERROR, FATAL

CPU 1

Resource Consumption - CPU + memory - return response - memory 256MB
- network throughput - " " - 100 MB/s

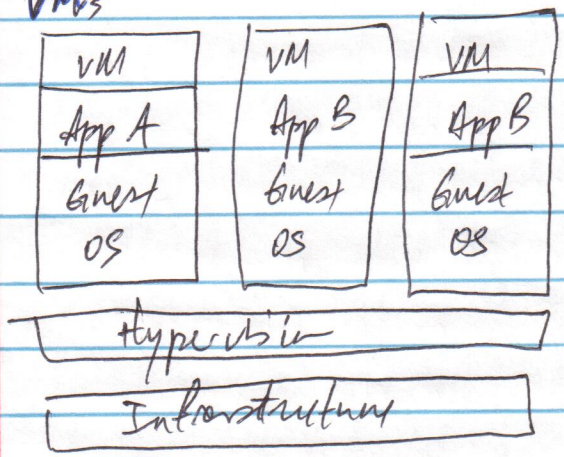
Maintence
- Extensibility - microservices
- Flexibility - monoliths

Anaphora Applicat's
- split functionalities
- merge "
- replace "
- state "

2.3 Container Orchestrati ō Kubernetes

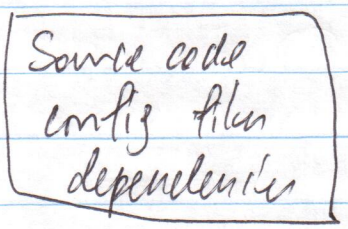
Cons
- declar for applicat packaging
- contain orchestrati ō Kubernetes
- Kubernetes resources
- declarative Kubernetes manifests

VMs



- standardizat.
- multiple VMs on same physical machine.
- replicated OS

- hypervisor - manages VM
- provides virtualizat of infrastructure



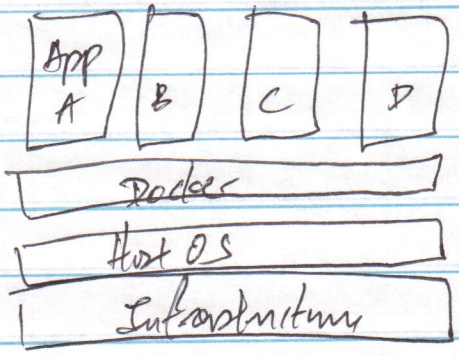
Docker
containers manager

Docker file
Docker Image
Docker Registry

Docker File – set of instructions
- create docker image
- optimum to package code + dependencies
- layers

- FROM - sets base image
RUN - execute command
COPY & ADD - copy files from host to container
CMD - set default command to execute when container starts.
EXPOSE - expose application port.

Containers



- Docker - container management tool
- lightweight
- better usage of resources

Docker image - read-only template
"create a run-able instance
- docker build [OPTIONS] PATH
  -- tag - name of tag
  -- file - name dockerfile

- dock run [OPTIONS] IMAGE [COMMAD] [ARG...]
  -- detach - run in background
  -- publish - expose container pnt to host
  -- it - interactive shell
  e.g docker run -d -p 5000:5000 python-hellowald

- docker logs {container_ID}

Docker registry - store + distribute
- tag - version control
  docker tag SOURCE_IMAGE[:TAG]
  TARGET_IMAGE [:TAG]
- e.g docker tag python-helloward repo-name/python-helloworld:v1.0.0
  repo name → image name; v1.0.0 → version (tag)

- docker push NAME [:tag]

eg. docker push repo-name/python-helloworld:v1.0.0
- OCI (Open Containe Initiative) compliant images
  - standardize images

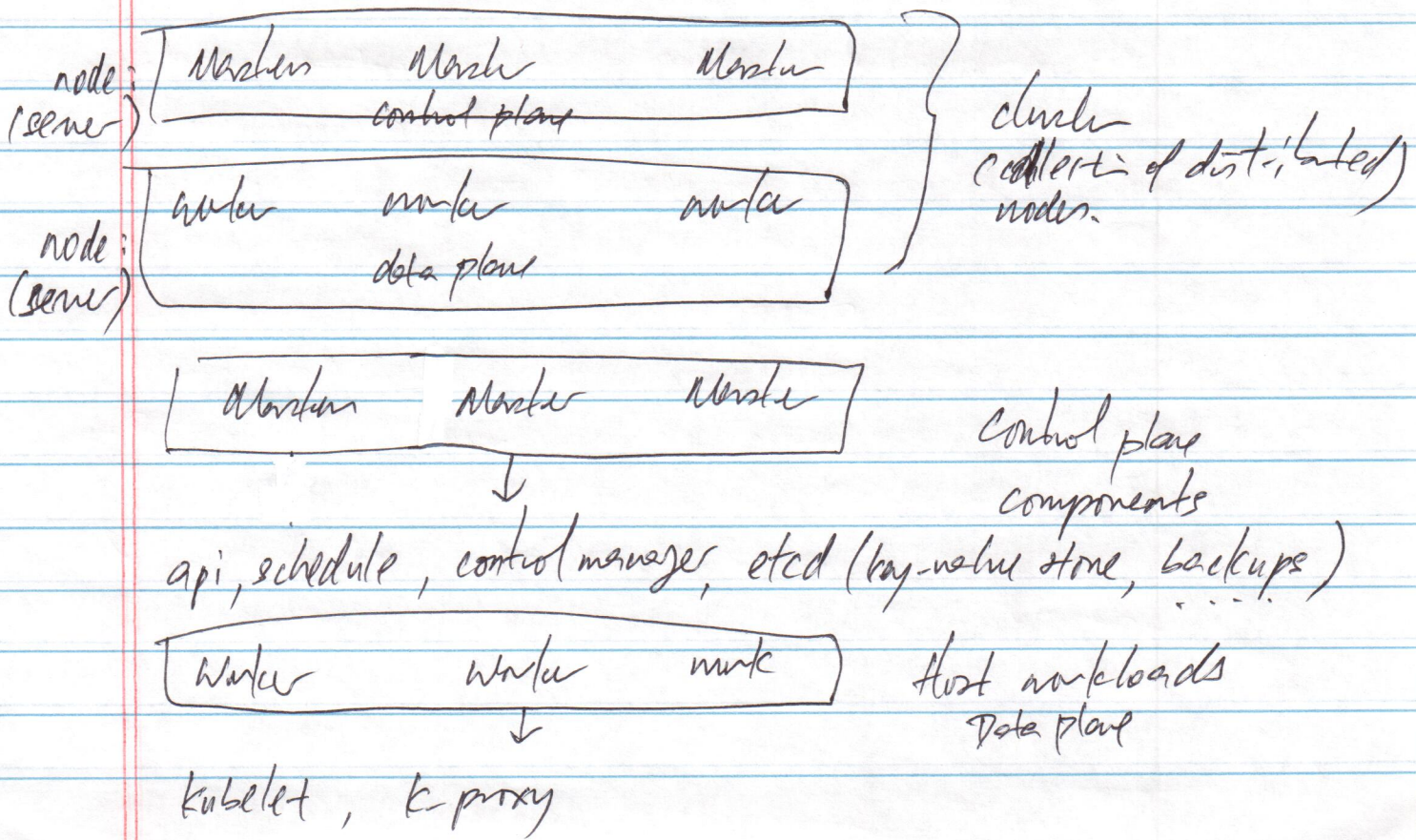
- Buildpacks - without Dockerfiles, do not expose Docker sockets
- Podman
- Buildah

Dockfile - FROM python:3.8
LABEL maintainer = "Name Here"
COPY . /app
WORKDIR /app
RUN pip install -r requirement.txt
CMD ["python", "app.py"]

go init foldername
docker ~~images~~ build -t python-helloworld .
docker images
docker run -d -p 5000:5000 python-helloworld
docker ps
→ local browser 127.0.0.1:5000
docker tag python-helloworld repo-name/python-helloworld:
docker push repo-name/python-helloworld: v1.0-0
v1.0.0

docker login
docker pull repo-name/python-helloworld:v1.0.0

Kubernetes - container orchestrator
- portability - platform agnostic
- scalability - horizontal pod autoscaler (HPA)
- resilience - elasticity
  - automated remediation - ReplicaSet
    self-healing - Readiness
    Liveness probe
- service discovery - cluster-level domain name system (DNS)
  - distribute traffic to load balance
- extensibility - building blocks principle
  - custom resource definition (CRD)
- operation cost - powerful scheduler
  - only occupies infrastructure it needs
  - cluster autoscaler



kubelet – agent that runs on every node, notifies kube-apiserver that this node is part of cluster.

kube-proxy – a network proxy that ensures reachability + accessibility of workloads placed on this node.

kubelet + kube-proxy – installed on all nodes in cluster (master and worker nodes)
– keeps k-apiserver up-to-date with list of nodes in cluster + manages connectivity + reachability of workloads.

cluster creation – bootstrap – kubeadm
– kubespray
– kops
– k3s
} production

– kind
– minikube
} development

k3s – light weight
– 1 binary install
– operational 1-node cluster
– kubectl – query cluster resources

vagrantfile
vagrant up
vagrant status, open VirtualBox
vagrant ssh

curl -sfL https://get.k3s.io | sh
sudo su
kubectl get no

kubeconfig - access to cluster
- ~/.kube/config
/etc/rancher/k3s/k3s.yaml for k3s
- export KUBECONFIG

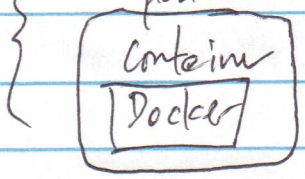
- cluster - metadata name, server, certificate-authority
- user - username, authentication - certificate-key
- token

- context - name, cluster, user
- links cluster to user

kubectl cluster-info - get cluster + add-on
" get nodes - get cluster nodes
" get nodes -o wide
" describe node {{ NODE NAME }}
PodCIDR, Pods, memory

kubernetes resources - pods - manages applications
- deployments + replicasets - oversees pods
- service + ingress - connectivity + reachability to pods
- configmap + secrets - passes configuration to pods
- namespaces - separation between multiple applications + their resources
- Custom resource definition (CRD)
extends kubernetes API to support custom resources.

- Pods - smallest manageable units in Kubernetes cluster.



1 pod : 1 container

1 node ≥ 1 pod

ReplicaSet - ensures desired amount of replicas for app is up + running.

Deployment → ReplicaSet → Pods

kubectl create deploy NAME --image=image [FLAGS]
--[COMMAND] [args]

- --replicas - set amount of replicas
- --namespace - set namespace to run
- --port - expose container port

e.g. kubectl create deploy go-helloworld
--image repo-name/go-helloworld:v1.0.0
--namespace test-space

Create pod:

kubectl run NAME ---image=name [FLAGS]
--[COMMAND] [args]

- --restart - restart [Always, OnFailure, Never]
- --dry-run [none, client, server]
- -it - open interactive shell to container

e.g. kubectl run -it busybox-test --image=busybox
--restart=Never

kubectl get deploy ⇒ deployment
kubectl get rs ⇒ replicasets
" " po ⇒ pods

kubectl edit deploy {{NAME}} -o yaml
kubectl port-forward {{RESOURCE}} {{PORT$}}

Rolling Update - updates pods in a rolling out fashion.
Recreate - kills all existing pods before new ones are created.
↳ minimum down time when upgrading.

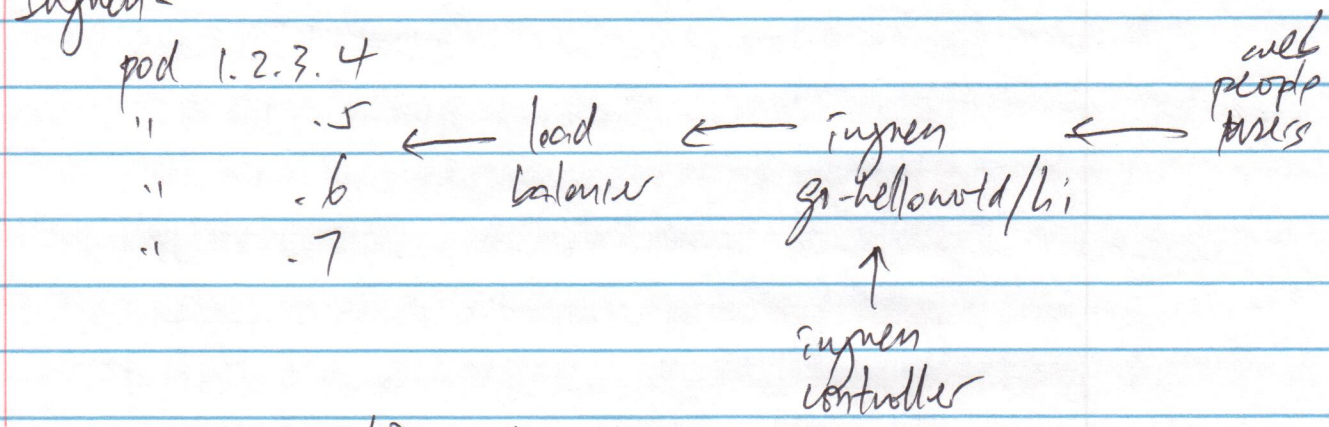
Services -
pod 1.2.3.4
pod 1.2.3.5 service:
pod " .6 ← abstract layer ← workload
pod " .7 10.103.44-200
pod " .8

(clusterIP - expose service to internal clusterIP.
NodePort - expose service using port exposed on all nodes in cluster.
LoadBalancer - expose services thru load balancer for public cloud provider.

kubectl expose deploy NAME --port=port [--target-port=port]
[FLAGS]
--protocol - set network protocol [TCP | UCP | SCTP]
--type - type of service [ClusterIP, NodePort, LoadBalancer]

e.g. kubectl expose deploy go-helloworld --port=81111
--target-port=6112

Ingress -



- exposes HTTP(S) routes to services within cluster using a load balancer provisioned by cloud provider.
- set of rules that map HTTP(S) endpoints to services running in cluster.
- Ingress controller keeps rules + load balancer up-to-date

kubectl get svc ⇒ services
kubectl run test-$RANDOM --namespace=default
--rm -it -- image=alpine --sh
wget -qO 10.105.166.228:6112
server IP

Configmap - non-confidential data
- consumed as environmental variable
Config file via a volume
command-line argument

kubectl create configmap NAME [flags]
--from-file - file path to key-value pairs
--from-literal - key-value pair from command line

e.g. kubectl create configmap test-cm --from-literal=
key=value

secret - sensitive data
- consumed as environmental variable
configfile via a volume

- kubectl create secret generic NAME [FLAGS]
--from-file
--from-literal } encrypt by base64

Namespace - logical separation between applications
- CPU + memory bound

kubectl create ns NAME
" get po -n NAME

kubectl get cm
" describe cm NAME
" " secrets NAME
" logs RESOURCE/NAME [FLAGS]
" delete "

Kubernetes manifests

- imperative
  - like
  - development
  - low learning curve

- declarative
  - files stored locally
  - production
  - high learning curve

YAML - data serialization standard

- apiVersion {{ API version }}
  kind: {{ Resource }} - labels:
  metadata: apps:
  spec: - container: - strategy name:
  template: image: namespace:
  name:

declarative way - kubectl apply -f deploy.yaml

template:
containers: — name:
ports: resources:
livenessProbe: requests:
readinessProbe: cpu:
memory:

kubectl create RESOURCE [REQUIRED FLAGS]
--dry-run=client -o yaml
kubectl apply -f manifest.yaml
" delete -f " "

- deployment (ReplicaSets)

Failure - readiness probe - restart
- liveness probe - pod is running, restart if needed
- service
- dep

- Control plane
  - if down, applications still works/runs
  - but can't start new workloads, no changes to existing ones
- Data plane
  - still runs if control plane is down

2.4 Open Source platform as a Service (PaaS)

Covers - PaaS mechanisms
- Cloud foundry
- Functions as a service

cluster management
- sandbox
- staging
- production

} x number of regions
- upgrade, update, manage, configure
- need a dedicated team in each region.

PaaS mechanisms
- cloud service - on-premise
- IaaS (Infrastructure as service)
- PaaS

Infrastructure services

IaaS
- AWS, GCP
- on-demand
- self-managed Kubernetes

- networking
- storage
- servers
- virtualization

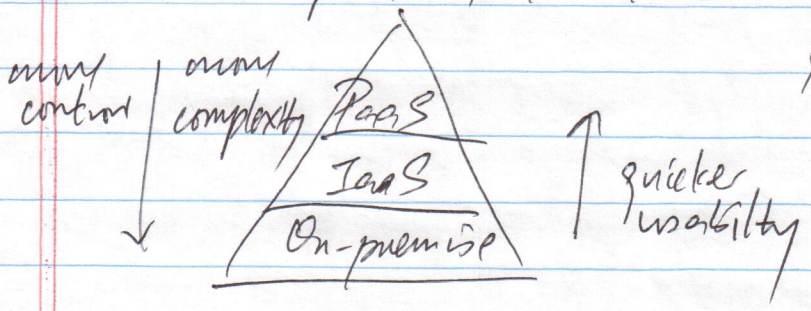
PaaS
- App Engine
- Beanstalk

- OS
- middleware (API, ..)
- runtime (JVM, ...)
- data (SQL, mongoDB)

- applications

on-premise - manage all layers on their own
- large engineering team

PaaS - Pros time efficiency - 3rd party handles management of infrastructure
- scalability + high availability
- rich application catalog
- Cons
- vendor lock-in
- data security - data located off-site
- operational limitations - limited to skills of provider



Cloud Foundry - open source PaaS
- no vendor lock-in
- handles application lifecycle - build, deployment, execution scalability
- deployed on top of Kubernetes
- routing - handles incoming external traffic + route to application
authentication - manages identities to use accounts
application lifecycle - controls application deployments, monitors status, handles changes
application storage + execution - handles artifacts to application

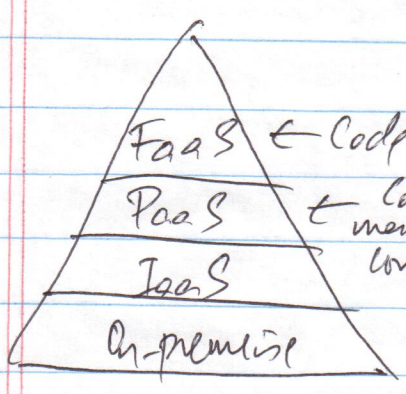
- services - use service brokers to provision on-demand dependency services for applicat (e.g database)
- messaging - communicat between Cloud Foundry components. routers + logging -

Kubernetes steps:
1. create OCI compliant image, created with Docker.
2. deploy Kubernetes cluster with a valid ingress controller for routing of requests.
3. deploy an observability stack, includes logs, metrics.
4. create YAML manifests for applicat deployment.
5. create CI/CD pipeline to push Kubernetes resources to cluster.

Cloud Foundry steps:
1. write manifest file to provide applicat deployment parameters.
2. deploy Cloud Foundry or PaaS solution from 3rd party.
3. deploy applicat to Cloud Foundry via CLI or UI
* cloud foundry will create OCI compliant image by default, provide routing capacities too.

Cloud Foundry - greater abstraction (less management of underlying infrastructure.)
- locks in PaaS vendor
Kubernetes - full control over contain orchestration, more flexibility in management of applicat.

Ephemeral services - run 1 to 2 times per day
- not cost efficient, billed even not in use
→ Function as a Service (FaaS)



{ event driven
- no management of infrastructure
- e.g. AWS Lambda, Google Function

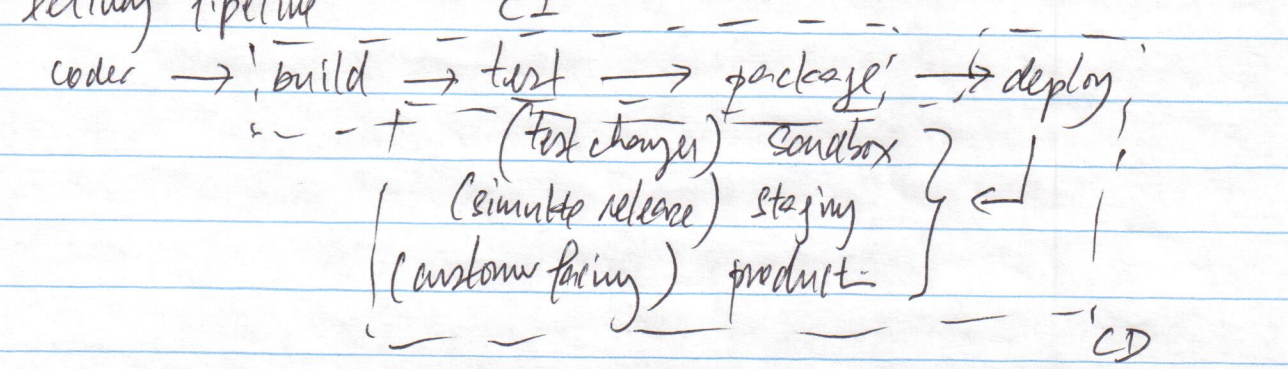
2.5 CI/CD with Cloud Native Tooling

Cover - CI/CD
- CI fundamentals
- CD fundamentals
- configmap managers - e.g Helm

Cloud Foundry
- simple
- automatically watch new commits
- one click redeploy

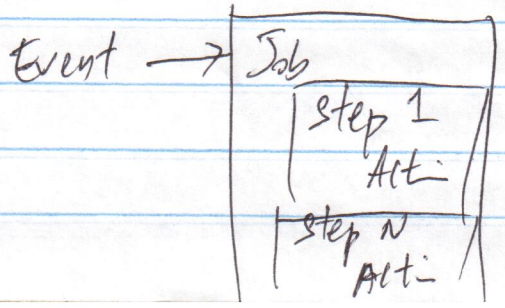
Kubernetes
- manual
- non-opinionated
- need to build a delivery pipeline

Delivery pipeline



Advantages of CI/CD
- frequency releases
- less risk - less manual intervention & configurations
- developer productivity - .

CI
- merge of code
- triggered at each commit
- output : Docker image ~ binary
- build : RUN pip install -r requirements.txt
  CMD ["python", "app.py"]
- test : pytest - functional tests
  pylint - static code analysis
- package : docker build -t python-helloworld
  docker push repo-name/python-helloworld:v1.0.0
- Jenkins, Circle CI, Concourse, Spinnaker, GitHub action
- GitHub action
  - event-driven - new commits
    - external event
    - scheduled events
  - multi-languages
  - notification
  - status badges



• .github/workflows/python-version.yaml

```
- name : Python version
  on : [push]
  jobs :
    check-python-version :
      runs-on : ubuntu-latest
      steps :
        - uses : actions/checkout@v2
        - uses :  "  /setup-python@v2
        - run : python-version
```

e.g. .github/workflows/docker-build.yaml

```
name : Docker build and push
on :
  push :
    branches : [master]
  pull_request :
    branches : [master]
jobs :
  build :
    runs-on : ubuntu-latest
    steps :
    - name : Checkout
      uses : actions/checkout@v2
    - name : Set up QEMU
      uses : docker/setup-qemu-action@v1
    - name : Set up Docker Buildx
      uses : docker/setup-buildx-action@v1
```

```
- name: Login to Docker Hub
  uses: docker/login-action@v1
  with:
    username: ${{ secrets.DOCKERHUB_USERNAME }}
    password: ${{ secrets.DOCKERHUB_TOKEN }}
- name: Build + push
  uses: docker/build-push-action@v2
  with:
    context:
    file: ./Dockerfile
    platform: linux/amd64
    push: true
    tags: {{ YOUR_DOCKER_HUB_REPOSITORY }}//
          python-helloworld: latest
```

Continuous Delivery (CD)

Push code to

1. sandbox
   - new changes tested
2. staging
   - release simulated

} automated

3. prodution
   - customer facing environment.
   - requires validation + triggering

CI → [sandbox → staging → prodution] CD

CI covers deploy stage
- imperative approach
  - kubectl create deploy python-helloworld
    --repo-name/python-helloworld:v1.0.0

- declarative approach
  - kubectl apply -f deployment.yaml

CI Tools:
- Jenkins
- CircleCI
- Concourse
- Spinnaker
- ArgoCD

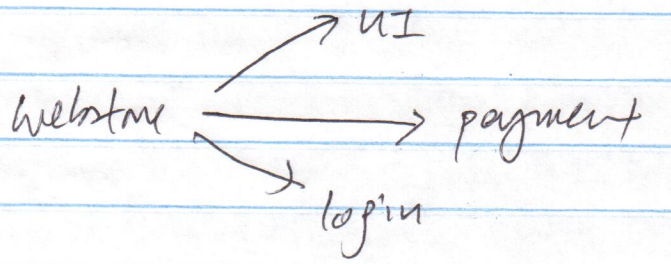
ArgoCD
- declarative CD for Kubernetes, follows GitOps.
- configurations stored in manifests.
- monitors new commits in Git repos.
- reconcile changes automatically or on manual trigger.

Advantages
- automatic deployment
- support for multiple config tools
- multi-cluster
- automatic reconciliations

Argo CD - Resources

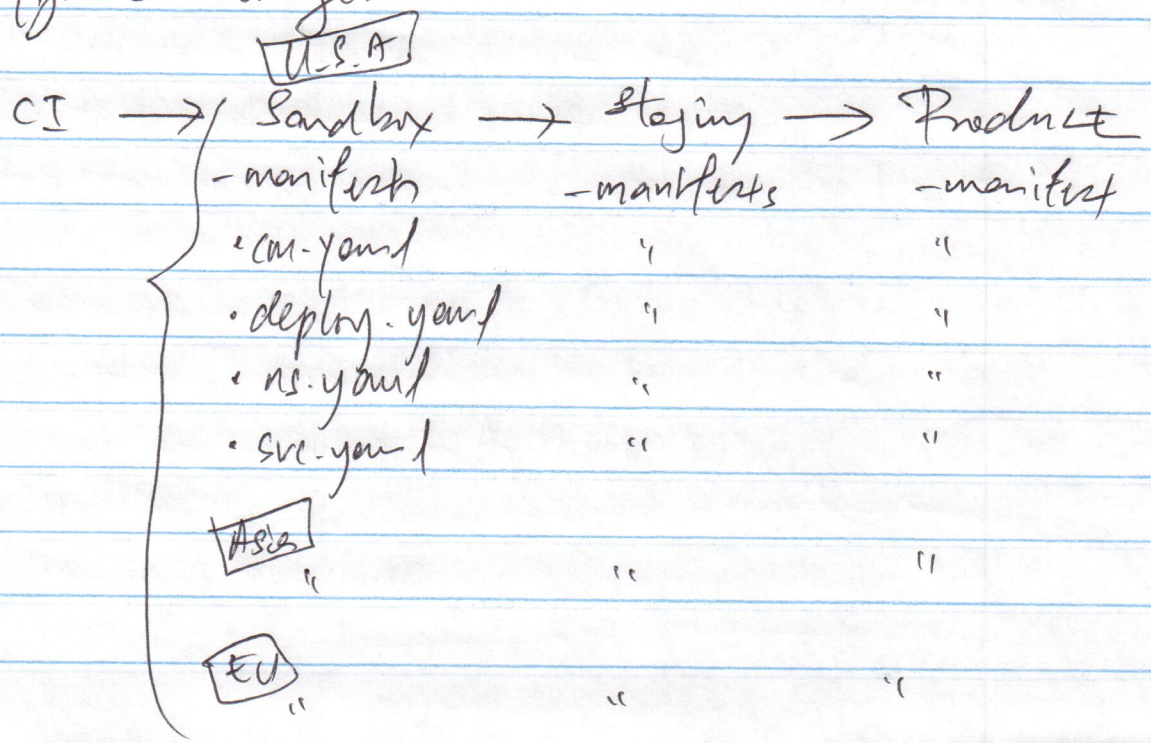
- Projects
  - Custom Resource Definition (CRD)
  - manage deployment of Kubernetes resources

- Application
  apiVersion:
  kind:
  metadata:
  spec:



App of apps:
- group of apps deployed + configured together.

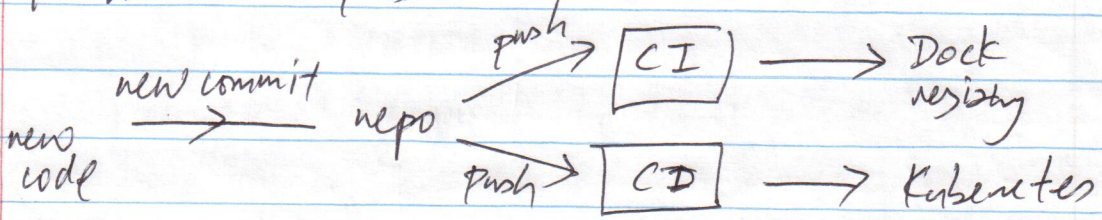
Configuration Managers



Tools:
- Helm - templates existing manifests + new files.
- Kustomize - template free, uses a base + multiple overlays.
- Jsonnet - programming language, templating of manifests as JSON files.

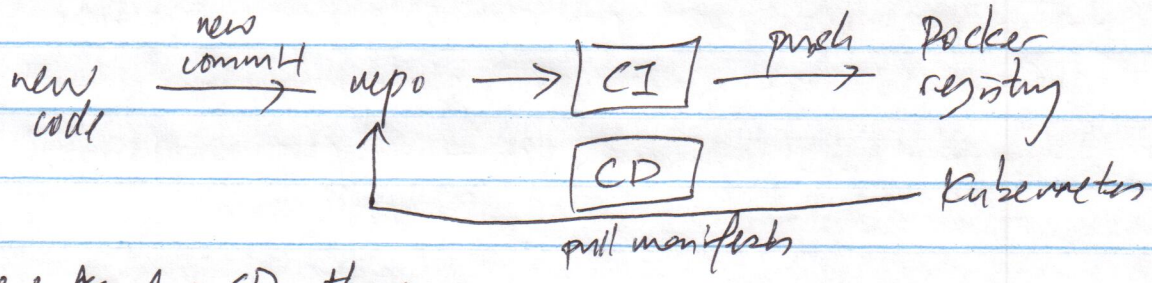
Helm:
- manages Kubernetes manifest through charts.
- Chart
  - collection of YAML files that describe state of multiple Kubernetes resources.
  - parameterized using GO
  - consists of:
    - chart.yaml
    - templates
      - configmap.yaml
      - deployment.yaml
      - namespace.yaml
      - service.yaml
    - values.yaml

Push-based CI/CD



e.g. Jenkins, CircleCI

Pull-based CI/CD



e.g. ~~Argo~~ ArgoCD, Flux.