Kubernetes - A tool for organizing, sharing, and managing containers. Gives devs the ability to scale, duplicate, push/rollback updates, VC

Pods - collection of containers

- Shared network namespace
- Shared storage volumes
- All containers in a pod co-located and co-scheduled on the same node
- Ephemeral
- Inter-container communication via localhost
- Sidecar pattern - enhance main container application without modifying main
- Proxy pattern - Intermediary between main and external world
    - Load balancing, caching, offloading responsibilities from main
- Adapter pattern - container to perform data format conversions to offload main resources
- Pod templates
- Pod affinity and anti-affinity - scheduling/restriction rules
- Pod autoscaling
- Pod eviction and disruption
- Pod health probes
- Taints and tolerations
- Pod are assigned a unique DNS

Services - abstraction layer over pods

- Services act as an intermediary between pods within the cluster
- ClusterIP service - communication between components in cluster
    - Eg: API layer and database layer talking frequently
- NodePort service - Makes API layer available on external ports
- LoadBalancer service
- ExternalName service - gives kubernetes DNS name
- Service discovery with DNS
- Headless services - direct p2p communication between pods - no DNS
- Service topology - user accesses nearest pods
- External traffic policy
- Sticky sessions - All individual user requests directed to same pod
- External databases

Deployment- application manager

- Deployments configure replica sets
- Have pod template specs
- Needs an update strategy - default is rolling updates
- Could also use declarative updates
- Easily scalable by adjusting number of replicas
- History/revision control
- Liveness and readiness probes

Clusters

- Control Plane - brain - API server, controller manager, scheduler, etcd - data storage

- On-premise cluster - local data center - bare metal control
- Public cloud managed cluster - cluster maintenance automatically done by provider
- Private cloud managed cluster -
- Local dev clusters - for testing on dev's local computer during dev phase of project
- Hybrid cluster - for security reasons
- Edge cluster - physically located close to user - for IoT devices - low latency
- High-performance computing cluster - large simulations or data processing - special hardware
- Multi-cluster federation - managing multiple clusters as if they were one

YAML file

- apiVersion: apps/v1
- Kind: Deployment, service, pod
- Metadata: name, labels, namespace
- Spec: desired state - containers, image, ports, volumes, restartpolicy
    - spec.replicas: This is the number of Pods you want to run.
    - spec.selector: This is how the Deployment identifies the Pods it should manage.
    - spec.template: This is the template for the Pods the Deployment creates.
    - spec.type: This defines the type of Service. Common types include ClusterIP, NodePort, and LoadBalancer.
    - spec.ports: This is where you define the ports the Service should expose.
    - spec.selector: This is how the Service identifies the Pods it should manage.
- ConfigMap - API object to store public data in key-value pairs
- Secret - API object for sensitive data like passwords - type and data fields

Scaling

- Horizontal - more nodes of same resources
- Vertical - more resources on the name number of nodes
- Multidimensional - diagonal scaling - doing both
- Elastic - automatically adjust based on demand - multidimensional

Security models

- Zero trust model - every user banned until whitelisted
- Shared responsibility model - outline different stakeholder responsibilities
    - Cloud provider responsible for the following
    - Infrastructure security - bare metal on sight security
    - Operational security - ensure service is online and not ddosed
    - Software supply chain security - enforce signature verification on new containers
    - You are responsible for:
    - Workload security - access control and encryption
    - Network security - connection between workloads have firewall and secure endpoints
    - Identity and access management - roles and perms
    - Software supply chain security - containers securely stored
- Use minimal base images - close all IP ports except those in use
- Regularly update and patch
- Vulnerability scanning - cloud provider services

- Use runtime security - sandbox containers at runtime - risk of container escape if not used
- Implement access controls
- Encrypt data
- Audit log activity
- Use binary authorization - only trusted images allowed in environment

Infrastructure as code - the YAML configuration files that dictate what machines should be

- Terraform - declarative syntax
- Ansible - agentless - relies on ssh and remote APIs - human readable YAML
- Puppet - agents - catalog based approach

Puppet

- Emphasis on declarative coding - defining desired state rather than procedural coding
- Idempotency - Operations executed many times will yield the same result regardless
- Test and repair - only act if testing deems it necessary to get to desired state
- Stateless - each puppet run is independent of the previous and next
- Facts - reports fact to puppetmaster and master responds with rules
- Resource type {attributes}
- Catalog - list of rules generated for node once facts have been input
- Modules - bundle configs and manifests
- Use templates to plug facts into in order to simply management of config files
- Nodes identified by FQDN - fully qualified domain names - webserver.example.com
- Puppet workflow => Client (agent) sends facts to the server (puppet master), which then processes the manifests for that node. Then generates catalog and sends back to client
- Uses Secure Socket Layer (SSL) - public/private keys
- Certificate Authority (CA) - Verified which public keys to trust
- Changing manifests will apply to all nodes in the fleet
- First test manifests on local then deploy
    - Puppet parser validate - check that manifest syntax is correct
    - Run using –noop parameter - See what would happen without actual
    - Rspec tests - set facts to different values and check that catalog should be good
- Canaries - First receivers that detect potential issues before they reach the other nodes

Monitoring

- Service Level Objectives (SLO) - like 99.9% availability - 3 nines
- SLA - Strict. Consequences

Troubleshooting

- Wipe VM and start new with old IAC
- Copy unhealthy VM to a known healthy node
- Read logs from log collection point
- Try raising an issue in a certain region
- Try the same system in a different machine type - more powerful
- Deploy the container not on an IAC node
- Need good backups and well documented recovery plan

DevOps - steps of dev lifecycle beyond writing code

- Continuous Integration - Constantly updating software

- Continuous Delivery - Testing changes and deploying as soon as verified
- Continuous Testing - in between build and deploy - unit, integration, system, smoke, load
- CI/CD, refers to the automation of an entire pipeline of tools that build, test, package, and deploy an application whenever a developer commits a code change to the source control repository.
- Feature flags - A/B testing for features
- Incremental rollout - slowly deploy small changes
    - Canary releases and blue-green deployments
- DevOps tools:
- Source code repositories, such as GitHub or Bitbucket
- CI/CD tools, such as Github Actions, Jenkins, and Google Cloud Deploy
- Infrastructure as Code (IaC) tools, such as Terraform or Ansible
- Container management tools, such as Docker or Kubernetes
- Security scanning tools, such as Snyk or SonarQube
- Production monitoring tools, such as DataDog or AppDynamics
- DevOps Lifecycle - Discover, plan, build, test, monitor, operate, continuous feedback
- Key performance Indicators (KPI) - performance metrics
    - Lead time for changes
    - Change failure rate
    - Deployment frequency
    - Mean time to recovery

Automation in DevOps

1. Remove manual steps
2. Reduce human intervention
3. Create consistency and reliability
4. Increase speed and efficiency
5. Increase scalability
6. Integrate and connect better
7. Reduce and handle errors
    a. Logging and flagging

Jira

- Value Stream Map - The workflow of producing deliverable
- Lead time, wait time, value added time
- Waste - Any time spent not creating value
    - Partially completed work
    - Extra features - out of scope
    - Relearning - lack of documentation
    - Handoff
    - Delays - refers to dependencies
    - Task switching - mental context switching
    - Defects - when bugs are released

DevSecOps

- Shift left security - Introduce security measures early on before writing most of the code
- Static Application Security tests

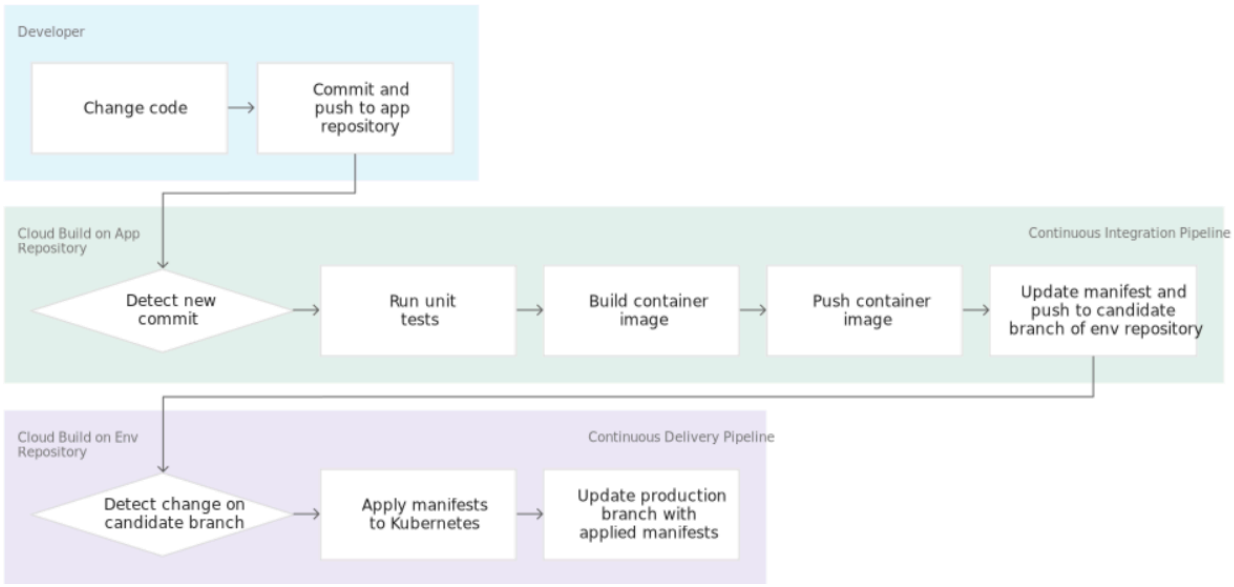
- Devs, cybersecurity, and IT pros working together
- Creating modular, isolated code

Release Management

- Release planning -scope, desired features, goals, timelines
- Versioning - v1.0 vs v1.1
- Coordinating dev and testing - managers work with testing teams
- Risk assessment and mitigation
- Communication and stakeholder mgt
- Release schedule - deadlines
- Change management
- Documentation and release notes
- Types of releases: Major, Minor, patch, hotfix

Postmortem

- Incident timeline: This describes what happened and when it happened.
- Root cause analysis: This includes details of why the incident happened.
- Impact analysis: This includes details of who and what was affected by the incident.
- Mitigation and recovery: This includes the steps taken to correct the incident.
- Action items for improvement: This describes the steps to take to prevent a future incident.


Developer
Change code
Commit and push to app repository
Cloud Build on App Repository
Continuous Integration Pipeline
Detect new commit
Run unit tests
Build container image
Push container image
Update manifest and push to candidate branch of env repository
Cloud Build on Env Repository
Continuous Delivery Pipeline
Detect change on candidate branch
Apply manifests to Kubernetes
Update production branch with applied manifests