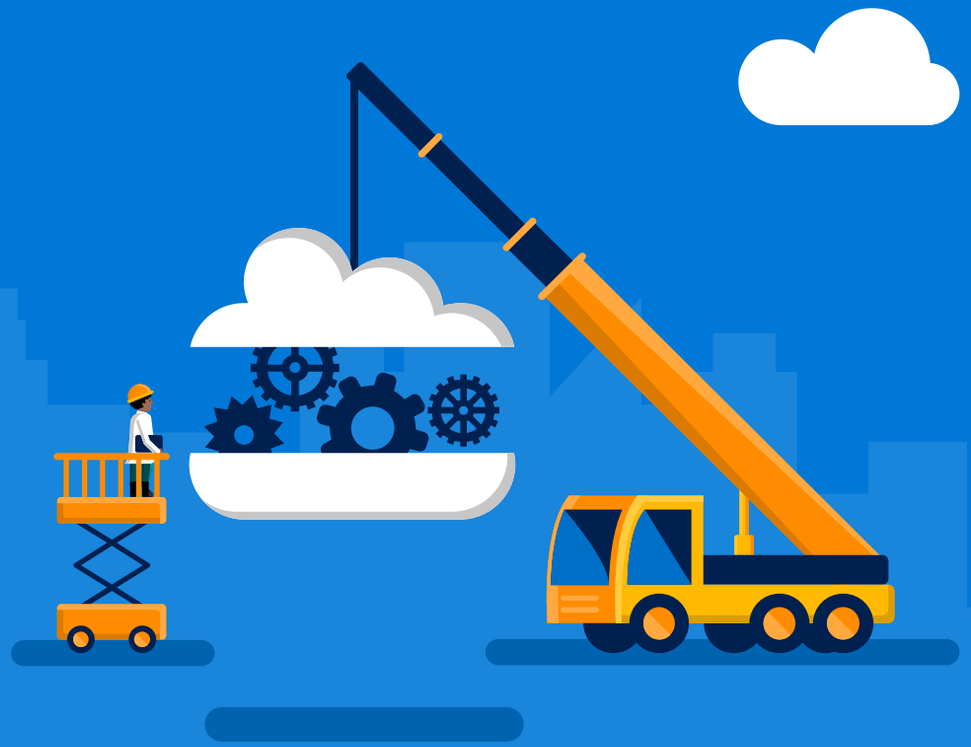
Microsoft

ARCHITECH

Kubernetes Bootcamp

Building Cloudy Applications

Jungho Kim
Jeevan Varughese



Hands-on Approach

- Go Deep into core areas – the fundamentals
- Go Broad in other areas, so you know what questions to ask
- Maximize hands-on time
- Deploy, reverse-engineer, break things, fix it
- Guided examples

How to get help...

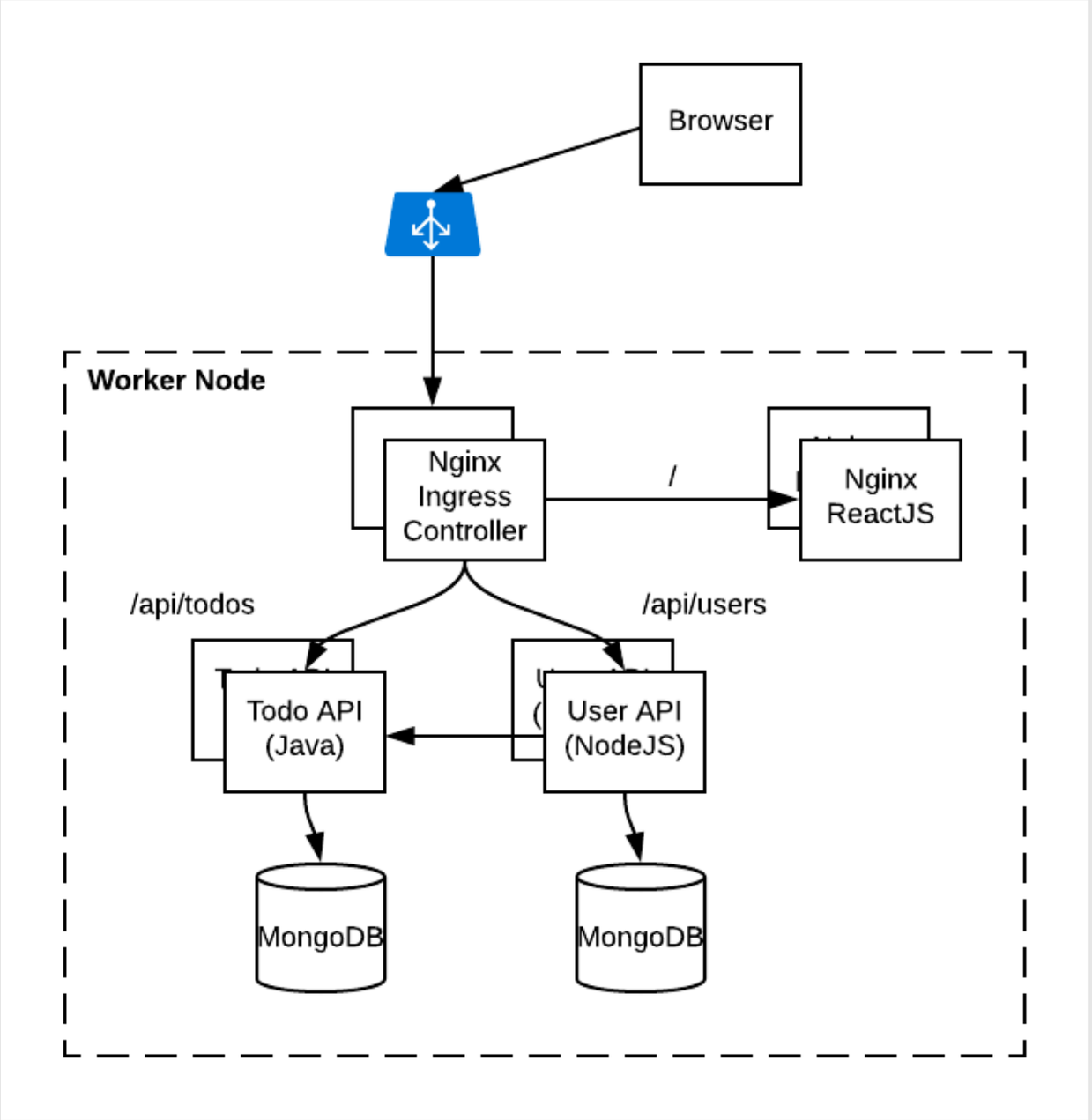
- Kubernetes API Reference: https://v1-10.docs.kubernetes.io/docs/reference/generated/kubernetes-api/v1.10/
- kubectl explain e.g. kubectl explain pods.spec, kubectl explain pods.spec.containers...
- Ask us :-)

Objective By End of Day

- You have hands-on exposure deploying a micro-service based application to K8S
- You have an understanding of K8S architecture
- You have an understanding of core K8S concepts and resources
- You have an understanding of K8S tooling
- You are an overview of the rapidly changing K8S space
- You are excited about K8S and how it can change the way you build enterprise solutions, and get product to market faster!

Todo Application

- Simple Micro-service based application
- ReactJS UI
- Java Springboot TodoItem API
- NodeJS User API
- MongoDB for the database
- Supports OpenID Connect/OAuth2 with Azure AD

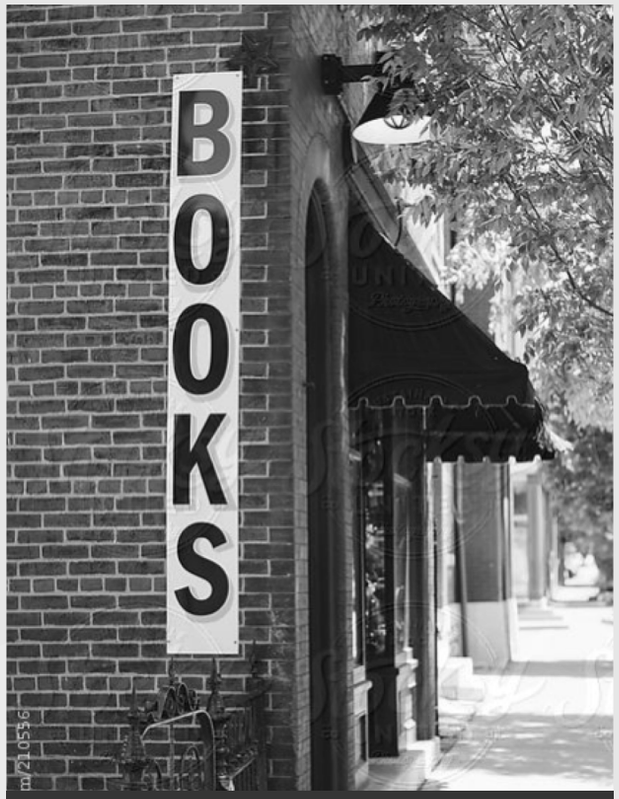


Why Modernize Apps?

NETFLIX

UBER

skype™



amazon.com

Our world is changing

Past

Long application cycles

Monolithic apps

Servers and VMs

Less data

Distinct infrastructure and operations teams

Today

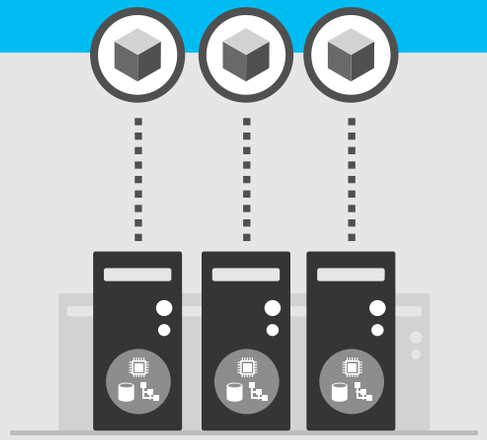
Rapid innovation

Loosely-coupled apps and microservices

Serverless

Big Data

Service-focused DevOps teams

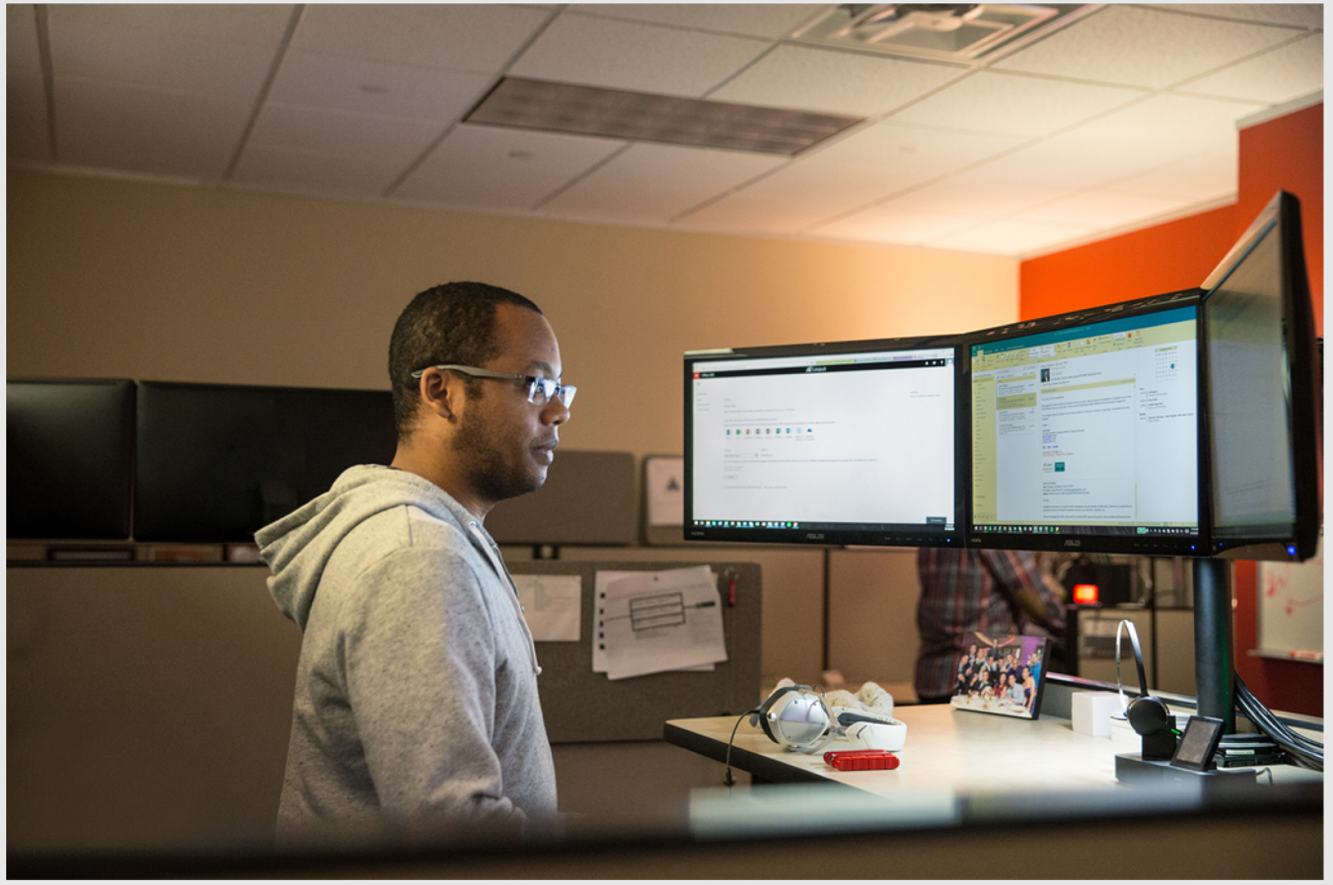


Servers

Services

Where do developers spend time…

- Maintaining ***existing*** applications
- "Fixing" ***existing*** applications
- Adding features to ***existing*** applications
- Fixing the feature someone else added to the ***existing*** applications
- Building a ***new*** application *(maybe … if there's time)*

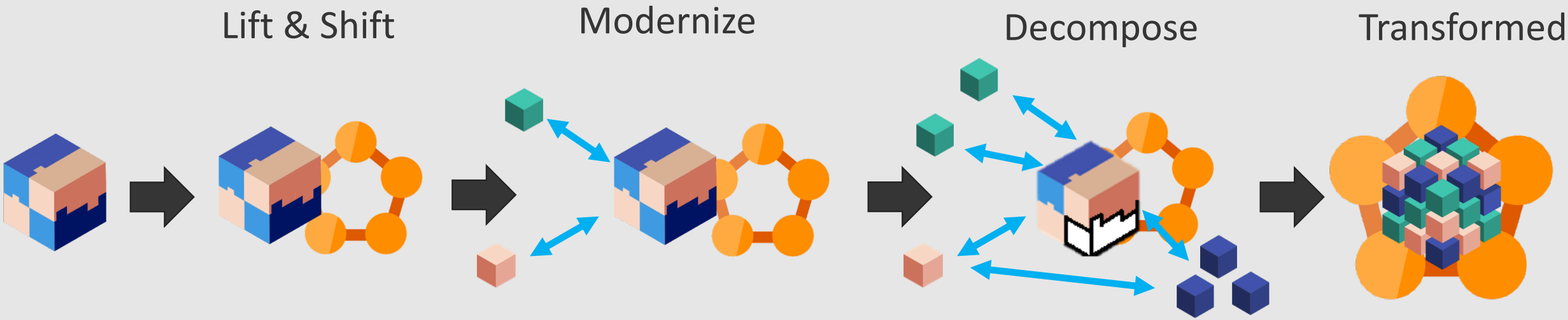


Where do application admins spend time…

- Maintaining ***existing*** applications
- Re-deploying new versions of ***existing*** applications
- Diagnosing issues with ***existing*** applications
- Troubleshooting issues from feature additions to the ***existing*** applications
- Deploying a ***new*** application

Help customers free resources and work more efficiently by modernizing traditional applications

Ways to Modernize Applications



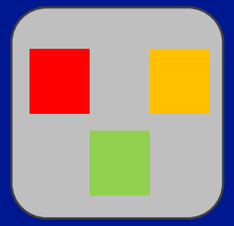
1) Traditional application

…You can stop at any stage

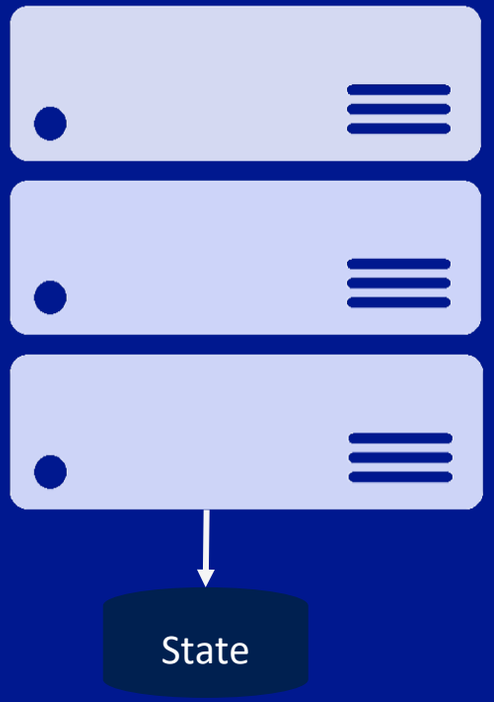
Monolithic application approach

- A monolithic application has most of its functionality within a single process that is commonly componentized with libraries.

Monolith



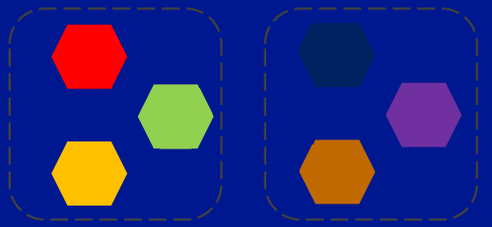
- Scales by cloning the app on multiple servers/VMs/Containers



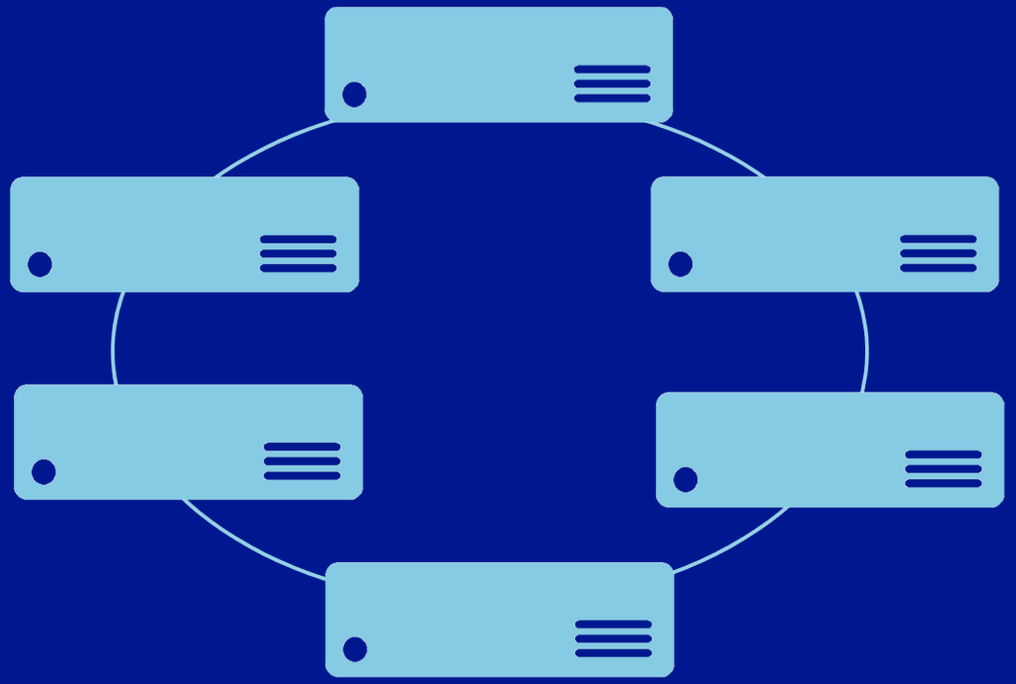
Microservices application approach

- A microservice application separates functionality into separate smaller services.

Microservices

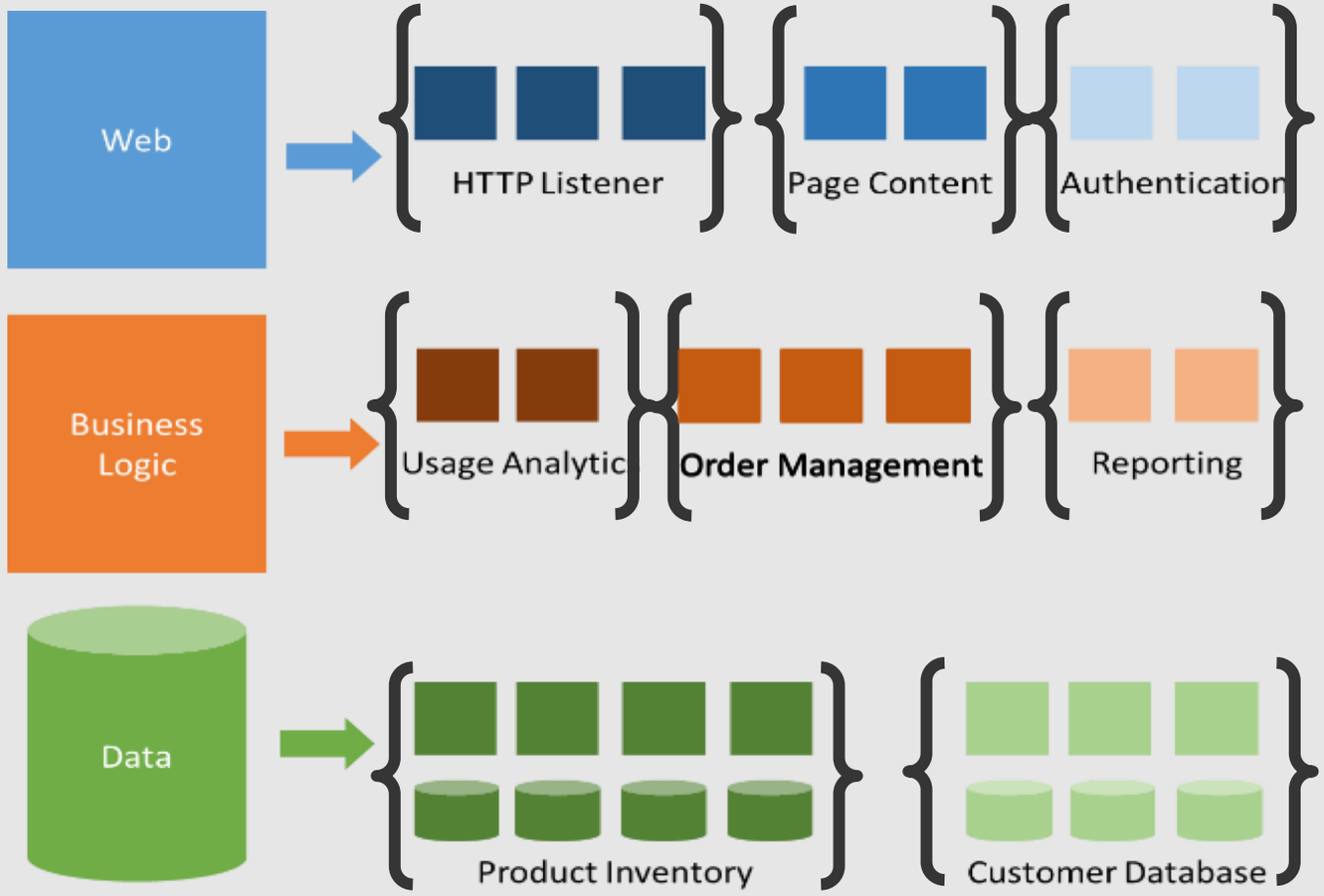


- Scales out by deploying each service independently creating instances of these services across servers/VMs/containers



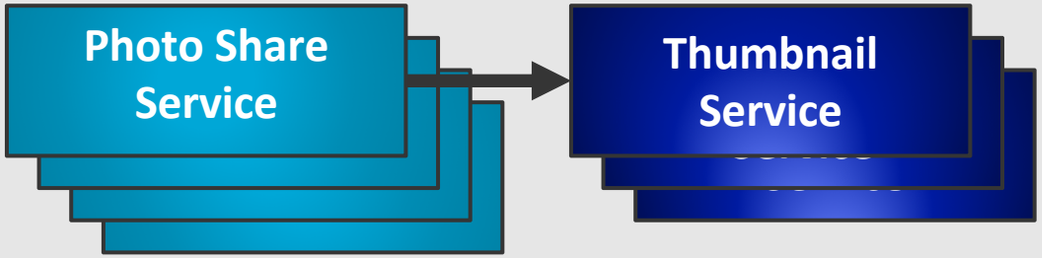
Modernization with microservices

- Individually built and deployed
- Small, independent services
- Integrate using published API
- Fine-grained, loosely coupled

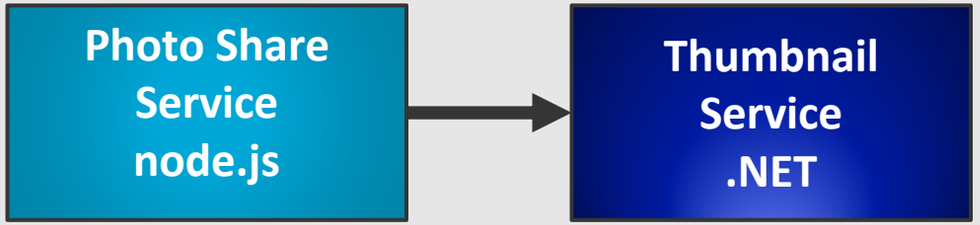


Microservice architecture benefits

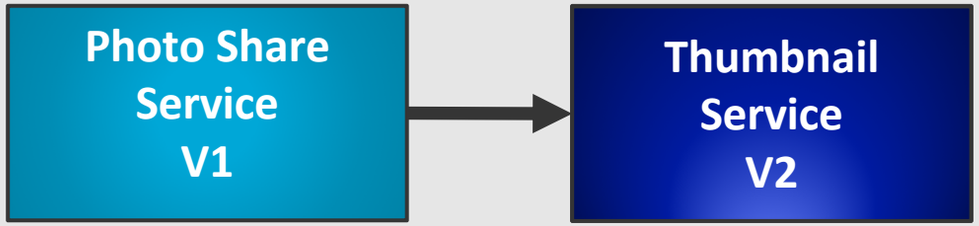
Scale Independently



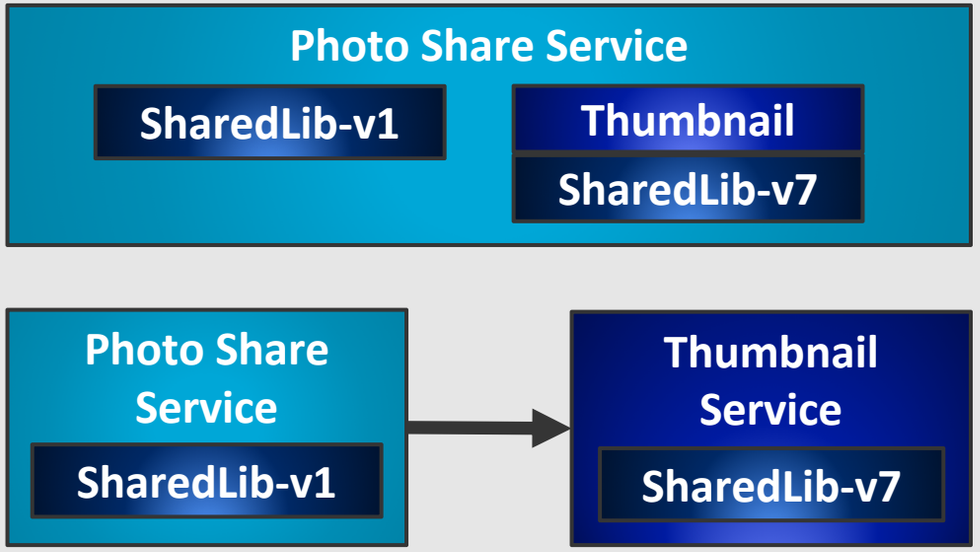
Different Technology Stacks



Independent Deployments



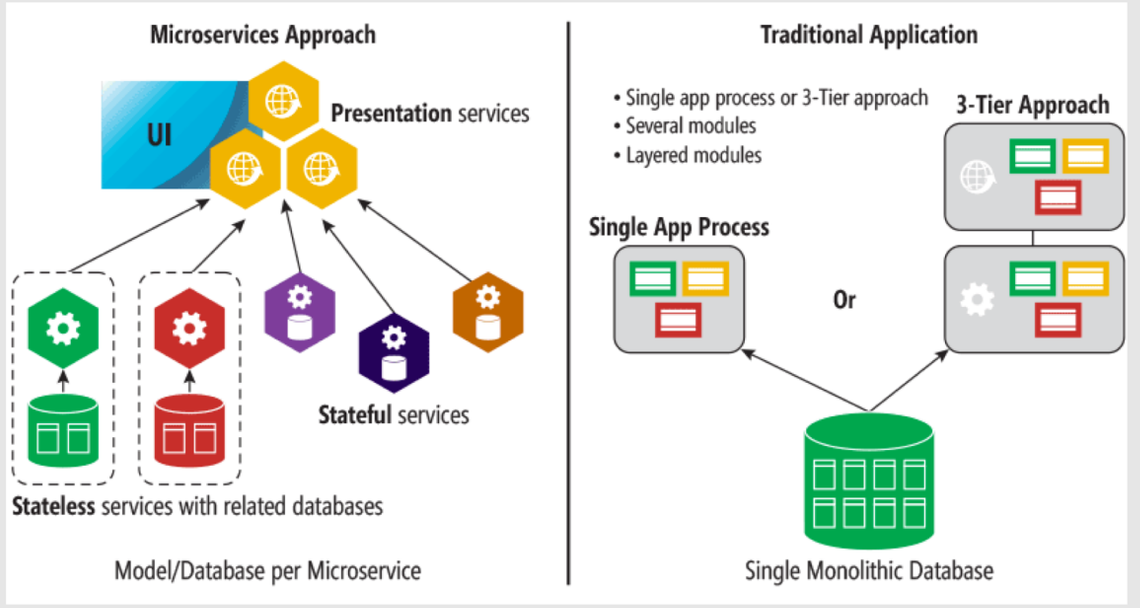
Conflicting Dependencies



Microservices Benefits

- ✓ Independent deployments
- ✓ Enables continuous delivery
- ✓ No downtime upgrades
- ✓ Improved scale and resource utilization per service
- ✓ Fault isolation
- ✓ Security isolation
- ✓ Services can be distributed across multiple servers or environments

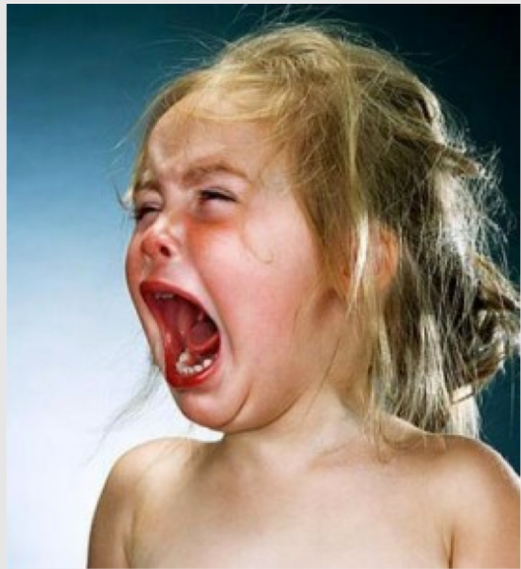
- ✓ Multiple languages / diversity
- ✓ Smaller, focused teams
- ✓ Code can be organized around business capabilities
- ✓ Autonomous developer teams



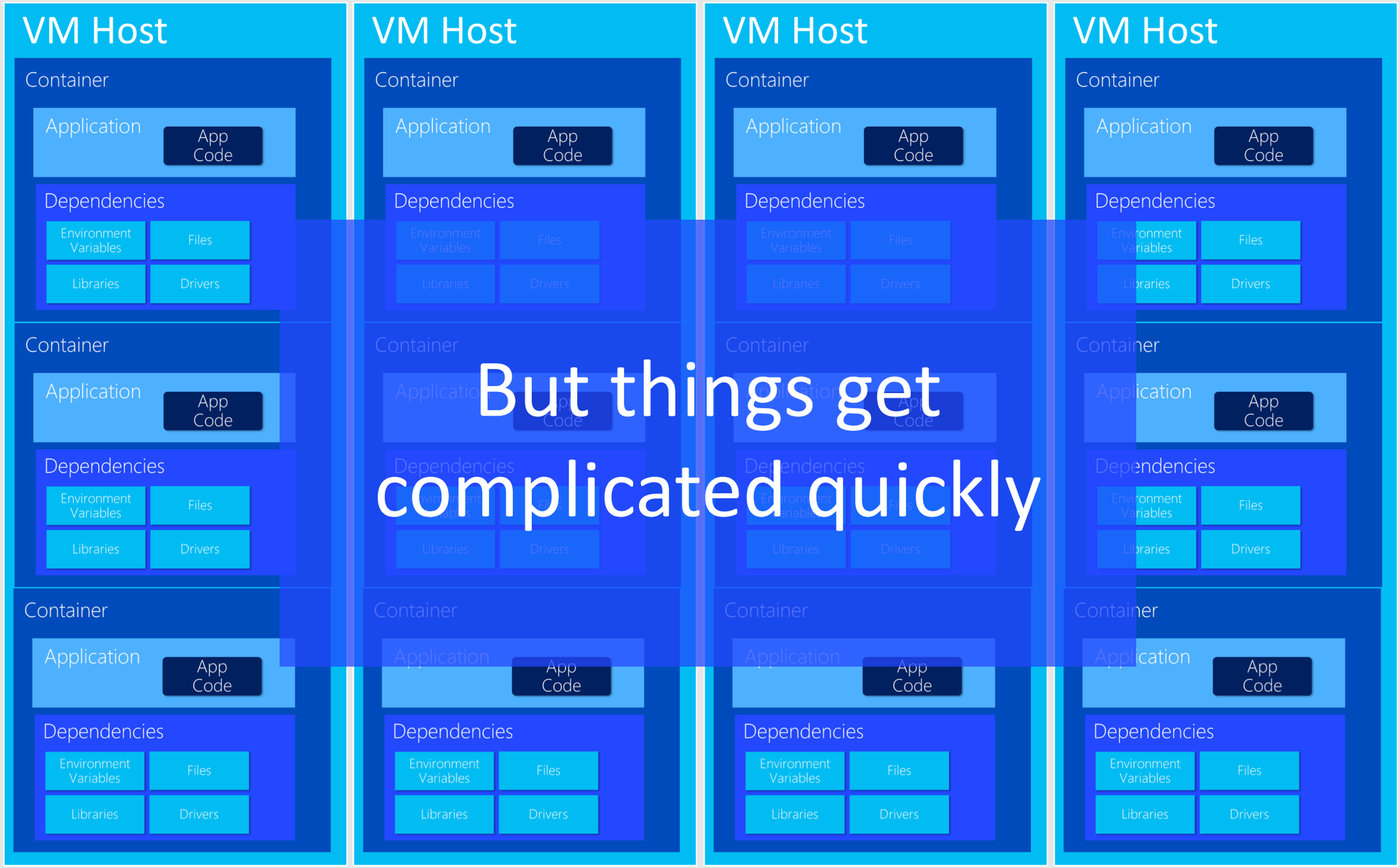
Microservices – The Hard Part

- ✓ Deployment is complex
- ✓ Testing is difficult
- ✓ Debugging is difficult
- ✓ Monitoring/Logging is difficult
- ✓ New service versions must support old/new API contracts
- ✓ Distributed databases make transactions hard
- ✓ Cluster and orchestration tools overhead

- ✓ Distributed services adds more network communication
  - ✓ Increased network hops
  - ✓ Requires failure/recovery code
  - ✓ Need service discovery solution
- ✓ Advanced DevOps capability: short-term pain for long-term gain



“Distributed apps are sufficiently complicated that they need to be flown by the instruments”


VM Host
Container
Application
App Code
Dependencies
Environment Variables
Files
Libraries
Drivers
But things get complicated quickly


VM Host
Container
Application
App Code
Dependencies
Environment Variables
Files
Libraries
Drivers
Very Quickly

What do cluster orchestrators do?

Provision	Security
Deploy (app / container)	Storage
Health Monitoring	Networking
Scaling	Service Discovery
Monitoring / Logging	Registry

Container Management at Scale

Cluster Management: deploy and manage cluster resources	**Scheduling**: where containers run	**Lifecycle & Health:** keep containers running despite failure	**Naming & Discovery**: where are my containers	**Load Balancing**: evenly distribute traffic
Scaling: make sets of containers elastic in number	**Image repository**: centralized, secure Docker container images	**Continuous Delivery**: CI/CD pipeline and workflow	**Logging & Monitoring**: track what's happening in containers and cluster	**Storage volumes**: persistent data for containers

Orchestrator Ecosystem (Constantly Changing)

Cloud services

- Azure Container Services (AKS, ACS)
- Azure Service Fabric (ASF)
- Amazon EC2 Container Services (ECS)
- Google Kubernetes Engine (GKE)
- Heroku
- Tutum

Low-level

- Docker Swarm
- CoreOS Fleet

Complete platforms

- Kubernetes
- Red Hat OpenShift (OCP)
- Cloud Foundry
- Mesosphere (Marathon)
- Enterprise DC/OS
- Docker Enterprise
- Rancher

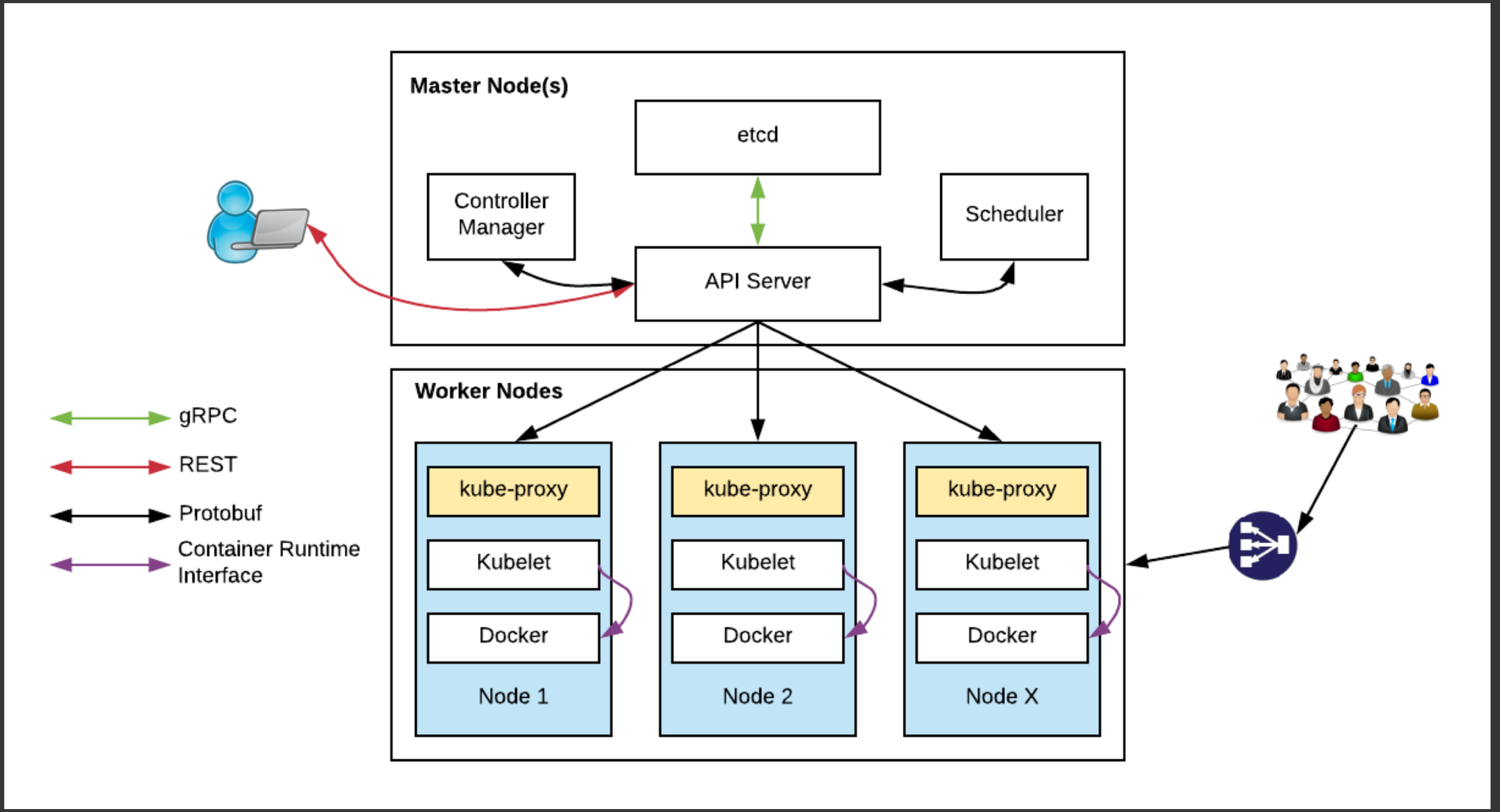
Why Kubernetes (K8s)?

- Industry Momentum
  - It’s Microsoft’s Direction
  - It’s Docker’s Direction
  - It’s OCP’s Direction
  - It’s Cloud Foundry’s Direction
- Brendan Burns
  - One of the Founders of K8s
  - Now works at MS heading up Container Strategies

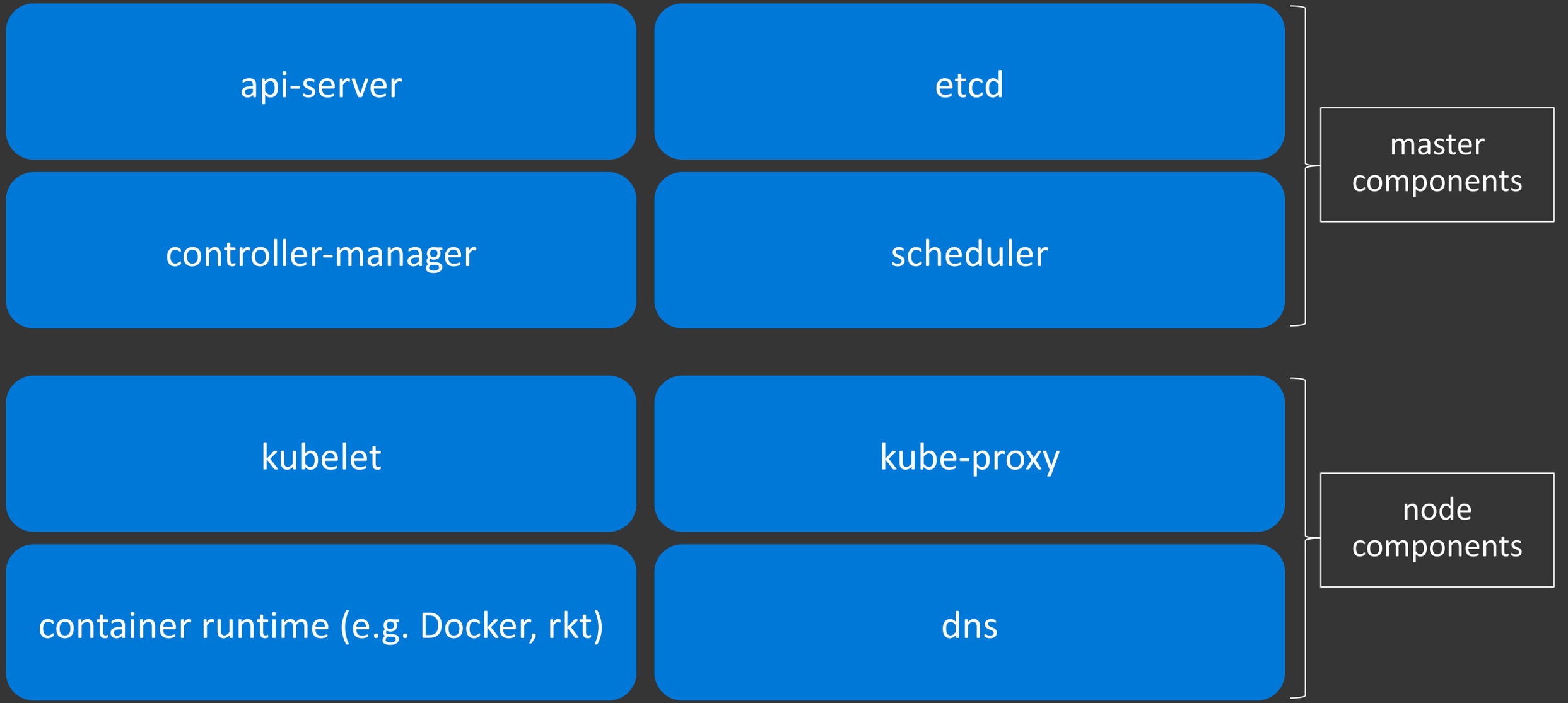
What is Kubernetes?

- History
  - Google open sourced borg. Google still actively involved, but less so every day
  - Schedules and runs application containers across a cluster of machines
  - Kubernetes v1.0 released on July 21, 2015. Joe Beda, Brendan Burns, & Craig McLuckie
  - Over 1,700 authors
- Key features
  - Declarative infrastructure
  - Self-healing
  - Horizontal scaling
  - Automated rollouts and rollbacks
  - Service discovery and load balancing
  - Automatic binpacking
  - Storage orchestration
  - Secret and configuration management
  - Not a PaaS platform

Kubernetes Architecture



Kubernetes Architecture Components



Master Components

- api-server: Front-end control plane. Exposes API
- etcd: Cluster database. Distributed, highly available
- controller-manager. Runs controllers, eg. replication controller, deployment controller, and more
- scheduler: assigns pods to nodes
- Add-ons
  - DNS
  - Heapster. enables monitoring and performance analysis
  - Dashboard
  - Logging

Worker Node Components

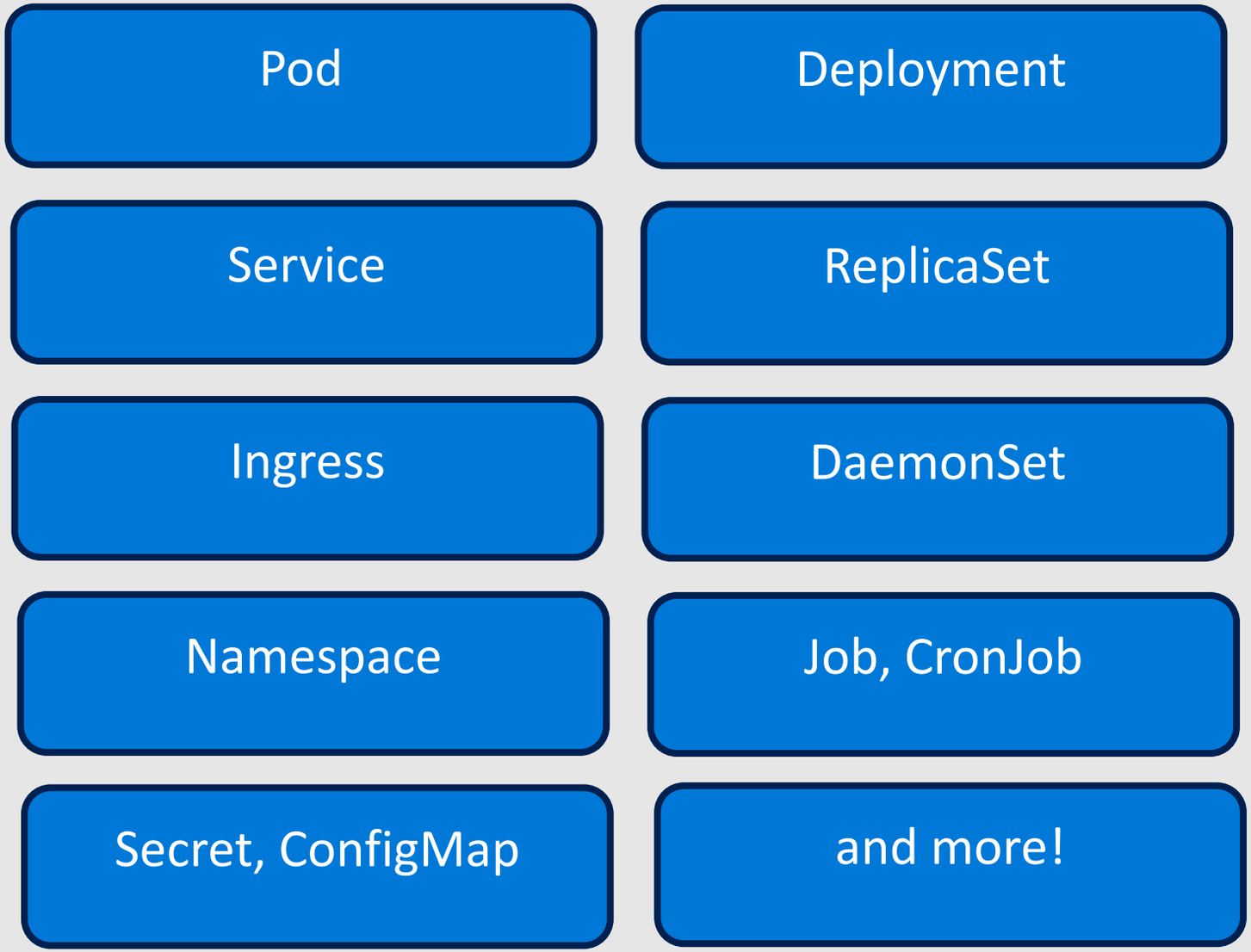
- kubelet:
  - Primary node agent
  - Watches and runs assigned pods
  - Executes health probes and reports status
  - Also key to “self-hosted” deployment of K8S where K8S is used to host K8S!!
- kube-proxy: enables network services
- docker: container engine (rkt supported experimentally)

kubectl: CLI to run commands against a Kubernetes cluster

- Swiss Army Knife: run deployments, exec into containers, view logs, etc.
- Syntax largely the same as `docker`
- Pronounced “koob sea tee el”…
  - Or “koob cuddle”

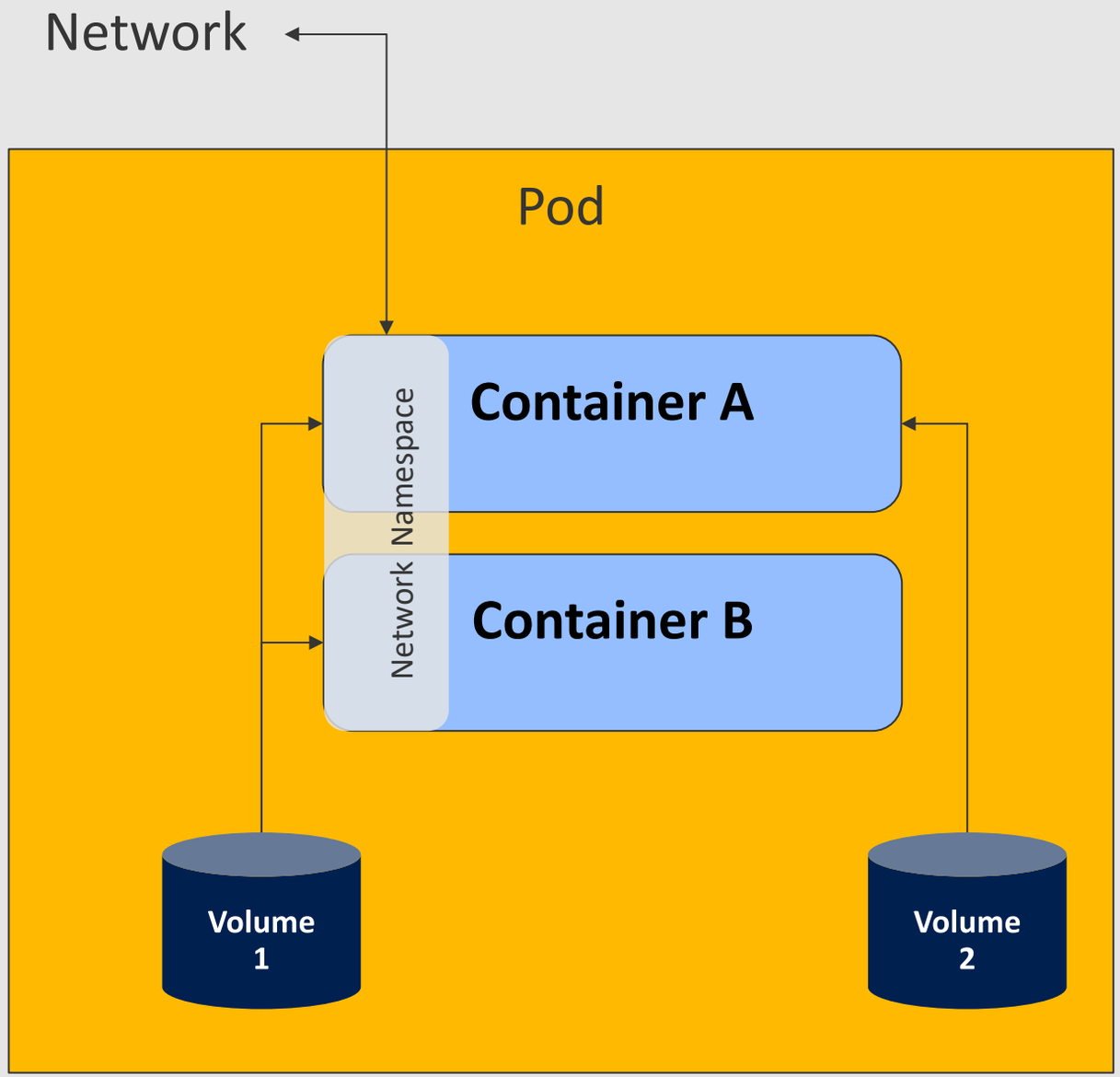
Kubernetes Resources

- Describes the ***"desired state"*** of your system
- Components such as Controllers, the Scheduler, API Server, kubelet work together to put the system into the "desired state"
- etcd stores this state



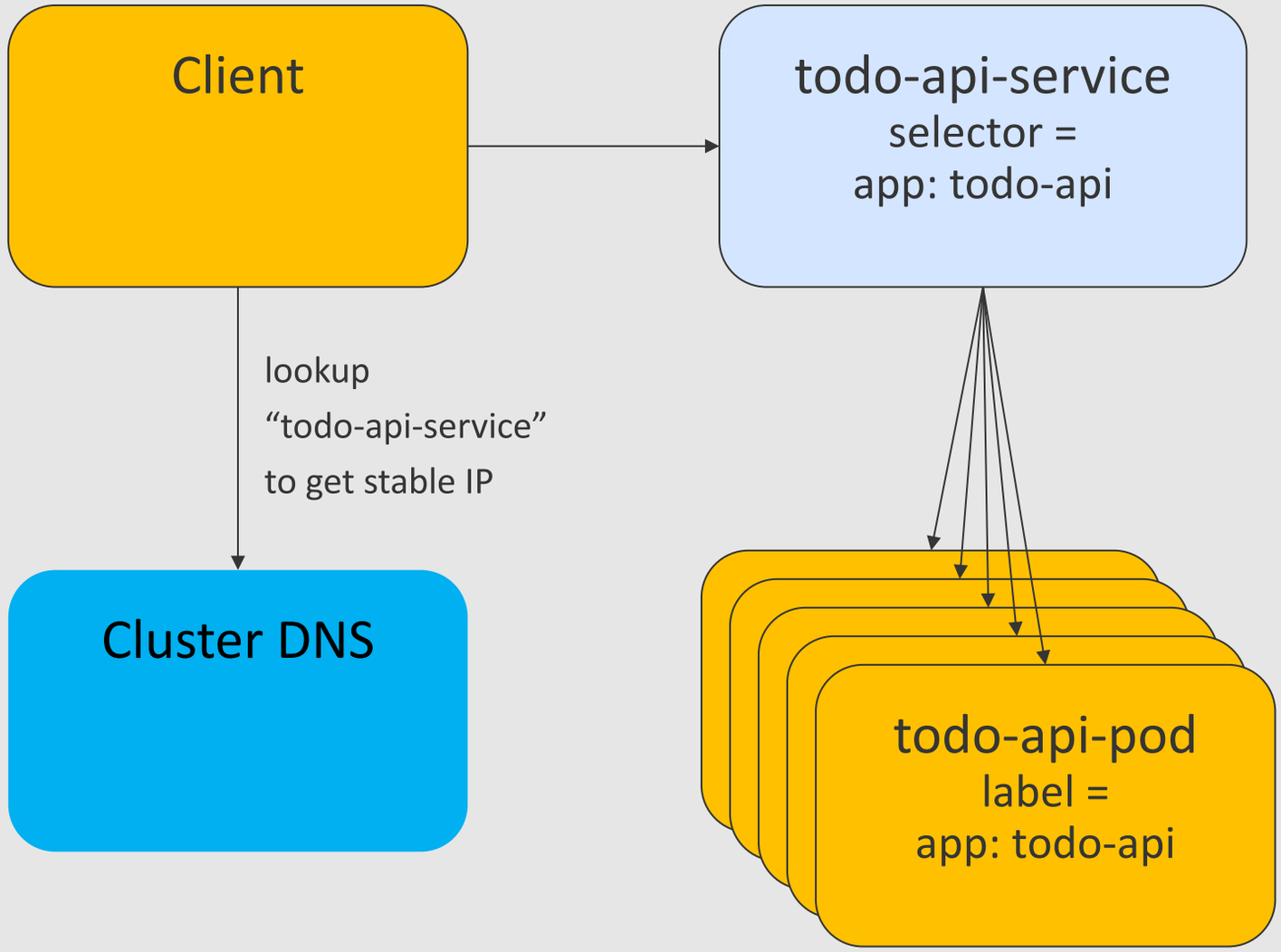
Pods

- Pods are the smallest unit of deployment in K8S
- Each pod gets an IP within the cluster address space
- They can container 1 or more containers
- The containers share a network namespace and can share volumes but have separate process namespace



Services

- Pods can come and go, so you need a stable IP to access your pods
- Services and cluster DNS enables Service Discover in K8S
- A service provides stable IP for clients
- A service manages the dynamic endpoints that represent the Pods IP

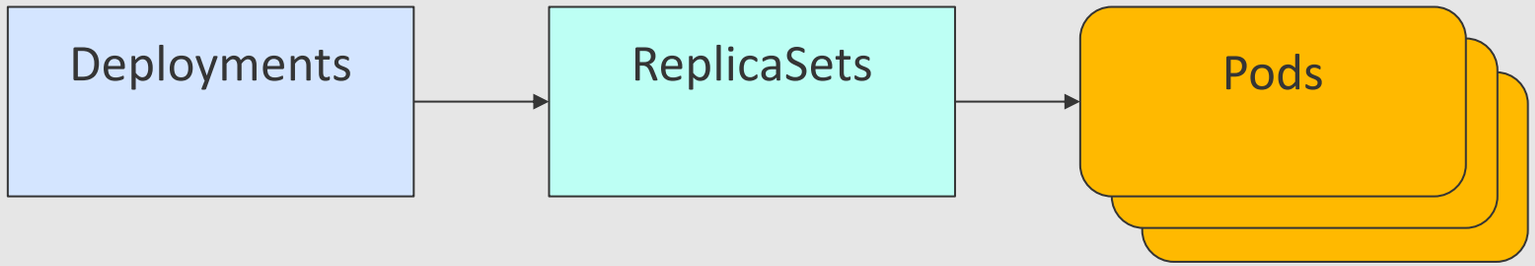


Services

- 5 types of services
  - ClusterIP
    - The default. Service IP is private to the cluster. Most of your services will be ClusterIP.
  - NodePort
    - Use when you have your own LoadBalancer
  - LoadBalancer
    - Cloud Provider provisions a LoadBalancer
  - ExternalName
    - Used to bind to external services (e.g. Your database that resides outside the K8S cluster)
  - Headless
    - A Service without an IP. Sometime, you want to know the IPs of the Pods, not the service, but you still want to do DNS lookup using a well-known name.

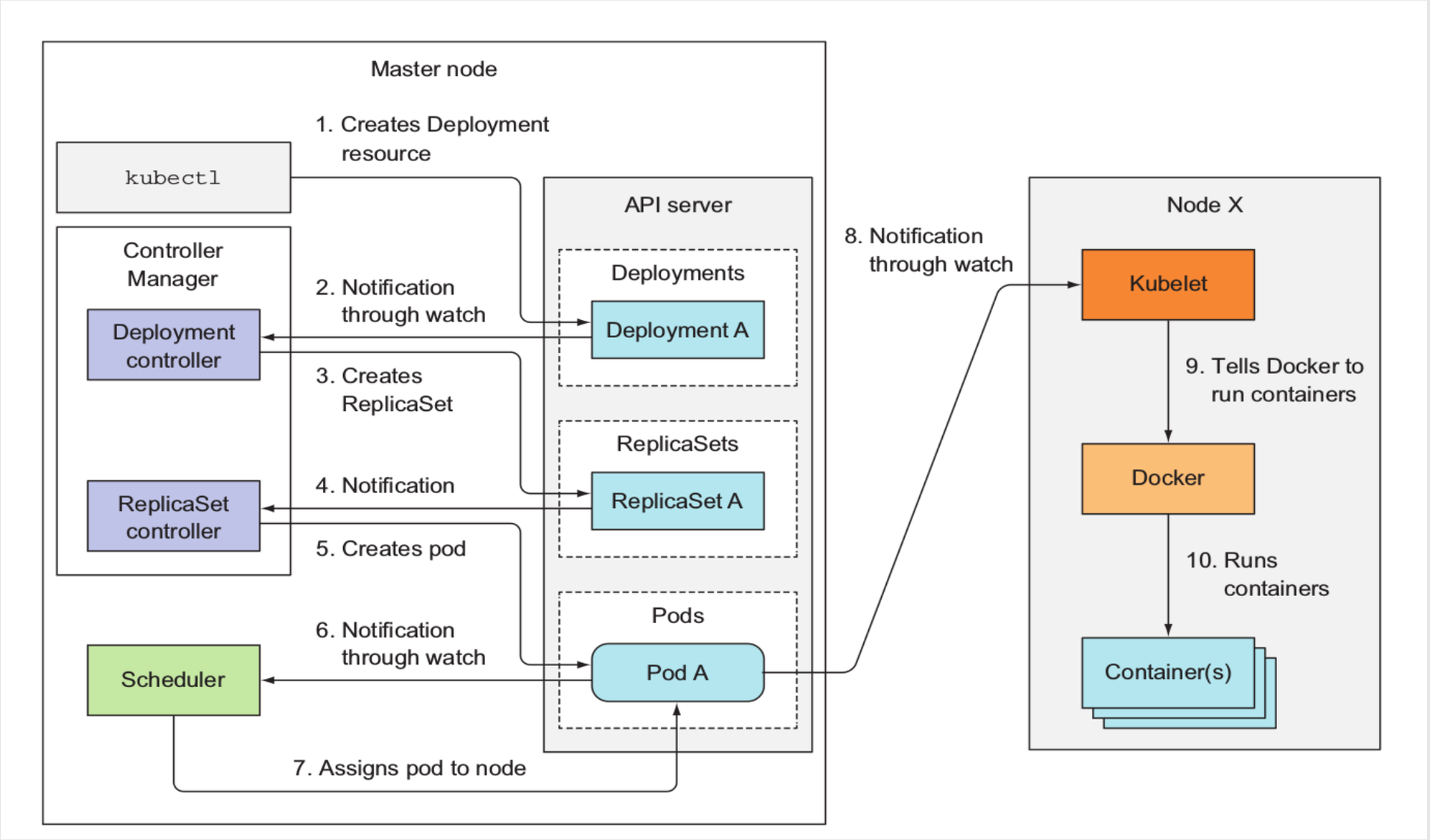
Deployments and Replicasets

- Deployments enable you to do controlled rollouts of your pods
- When you create a deployment, a replicaset is automatically created, the replicaset supervises many instances of your pods should be deployed




```
kubectl scale --replicas=3 deployment/todo-api
kubectl rollout status deployment/todo-api
kubectl rollout undo deployment/todo-api
```

Deployment Events



From "Kubernetes in Action" by Manning

Persistent Volumes

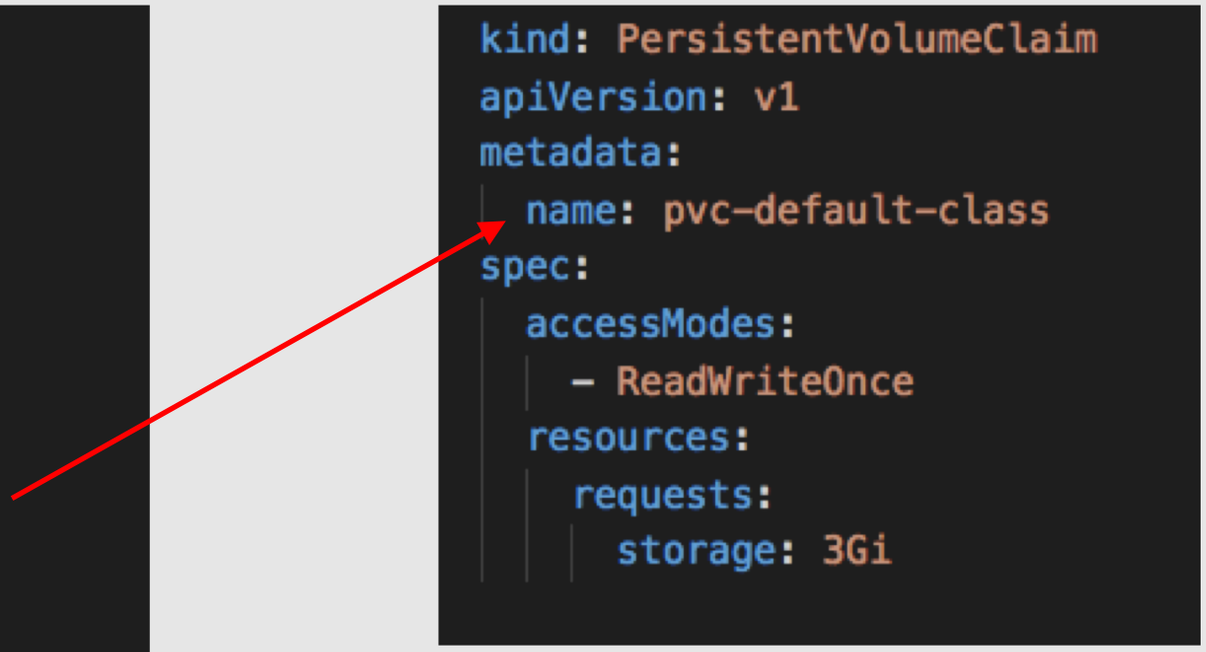
- Persistent Volumes are resources to specify how storage volumes are made available to pods
- The containers in your pods mount these volumes to a path in their filesystem
- K8S supports many volume types
  - https://kubernetes.io/docs/concepts/storage/volumes/

```yaml
kind: PersistentVolume
apiVersion: v1
metadata:
  name: task-pv-volume
  labels:
    type: local
spec:
  capacity:
    storage: 10Gi
  #only once process can read/write
  accessModes:
    - ReadWriteOnce
  #when the PVC is released, retain the data.
  #valid values are Retain (default), Recycle and Delete.
  #Not all volume plugins support Recycle. See the volume plugin for details.
  persistentVolumeReclaimPolicy: Retain
  # hostPath volume type so the volume is mapped to a path on the node
  hostPath:
    path: "/tmp/data"
```

Persistent Volume Claims

```yaml
kind: Pod
apiVersion: v1
metadata:
  name: task-pv-pod
spec:
  volumes:
  - name: default-class-pod
    persistentVolumeClaim:
      claimName: pvc-default-class
  containers:
  - name: nginx
    image: nginx
    ports:
    - containerPort: 80
      name: "http-server"
    volumeMounts:
    - mountPath: "/usr/share/nginx/html"
      name: default-class-pod
```

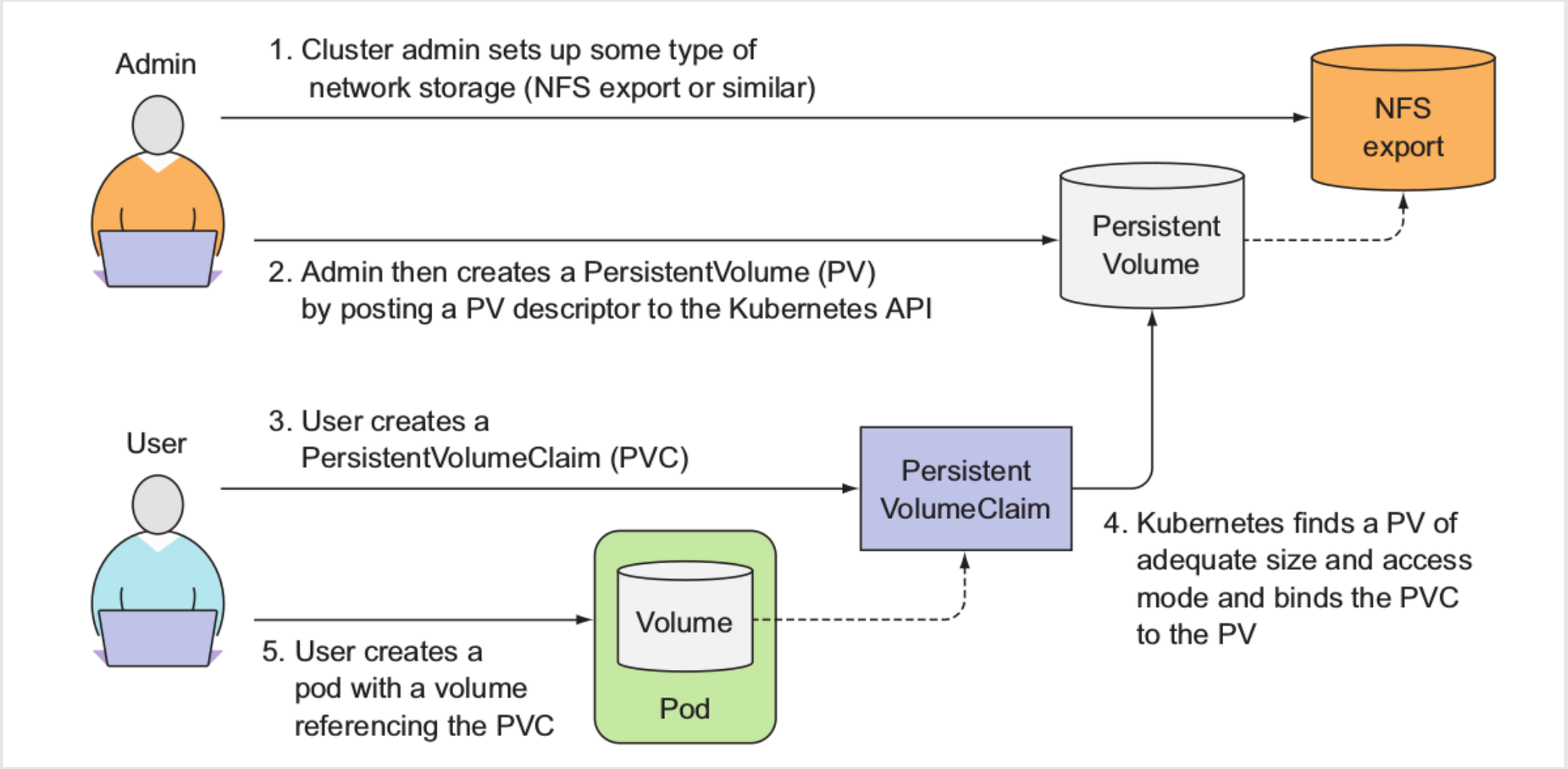
```yaml
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: pvc-default-class
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 3Gi
```



PVC to PV is 1-to-1

Pod to PVC can be *-1 depending on access

Persistent Volume Claims



From "Kubernetes in Action" by Manning

Static vs Dynamic Provisioning

- If the cluster admin creates a PersistentVolume, this is static provisioning
  - The cluster admin needs to make decisions about the size of the volume
- Persistent Volumes can be provisioned dynamically based on what is requested in a PersistentVolumeClaim
- This requires StorageClasses

Storage Class

- Storage classes are means for cluster admins to define different classes of storage for developers to use e.g. disk speed, capacity, etc
- Developers can then request storage of a specific class in their PVC
- If a storage class is not specified, and there is a default, then it will be used.

```
kubectl get storageclasses

NAME                 PROVISIONER
default (default)    kubernetes.io/azure-disk
managed-premium      kubernetes.io/azure-disk
```

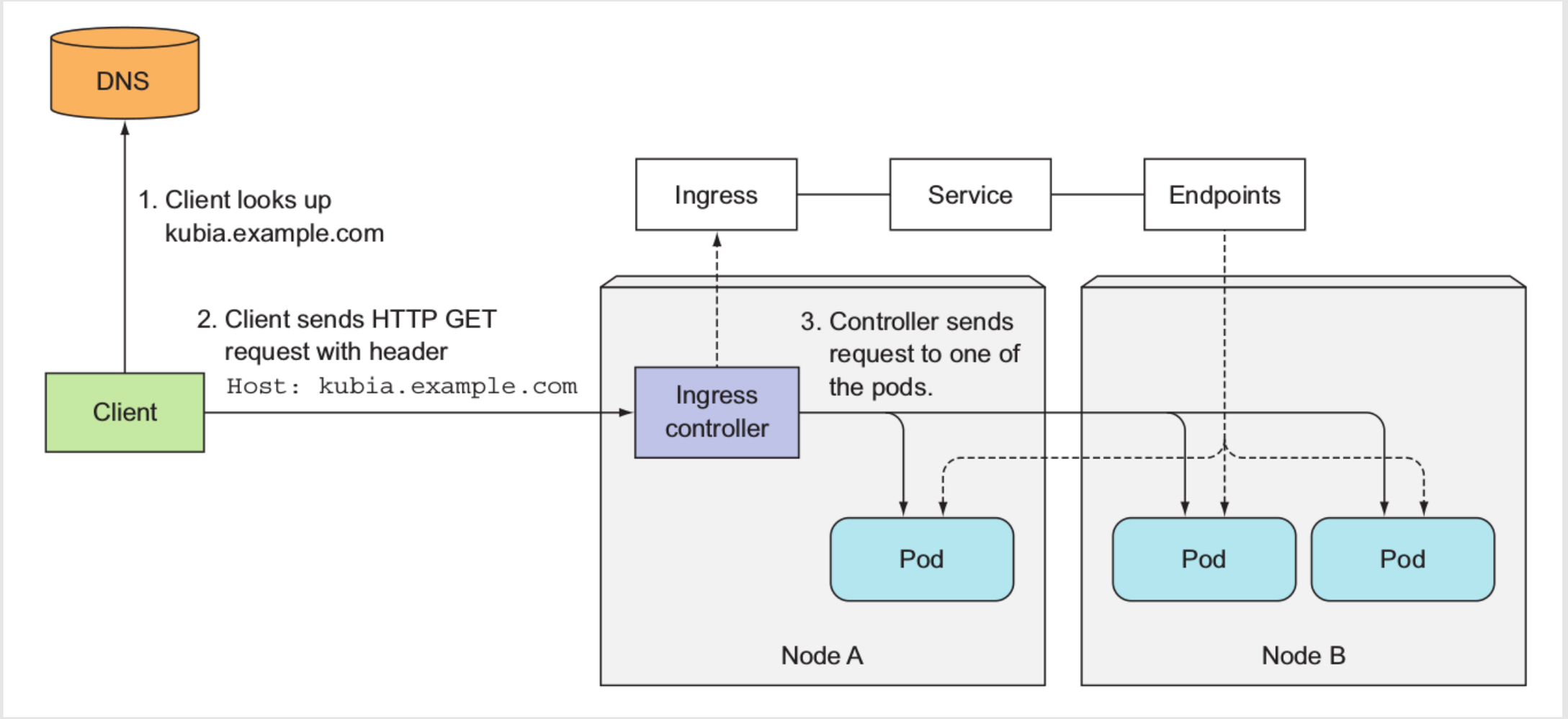
Default Storage Class for AKS

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  annotations:
    storageclass.beta.kubernetes.io/is-default-class: "true" ...
  labels:
    ...
  name: default
parameters:
  kind: Managed
  storageaccounttype: Standard_LRS
provisioner: kubernetes.io/azure-disk
reclaimPolicy: Delete
```

What user are you running as?

- The process within containers by default, run as root
- Much as possible, you do not want to run as root
- Problem: Volumes are mounted with root ownership
- Solution? Well, not so straight forward as it depends the volume type.
- Let’s see the examples in k8s-bootcamp/storage

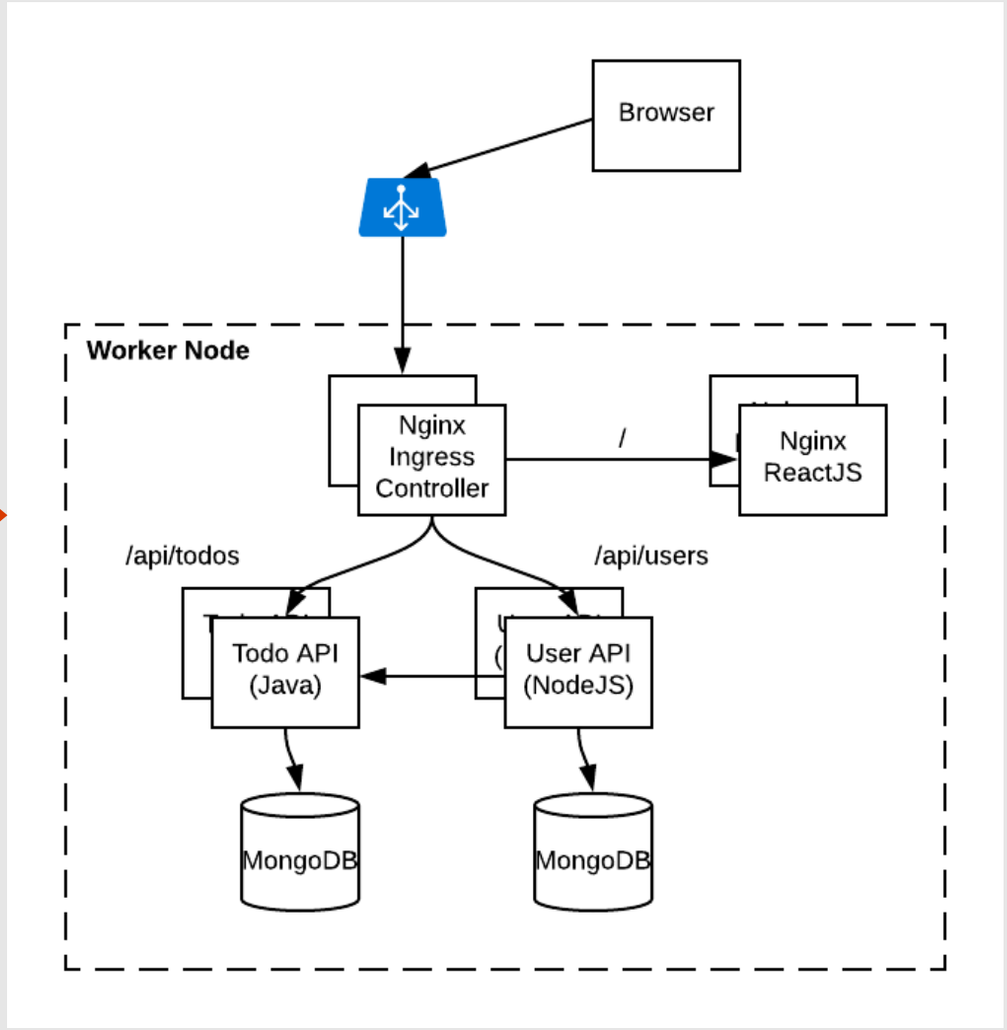
Ingress



From “Kubernetes in Action” by Manning

Ingress Rule

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: todo-app-ingress
  annotations:
    ingress.kubernetes.io/ssl-redirect: "false"
spec:
  rules:
  - host: 192.168.99.100.nip.io
    http:
      paths:
      - path: /
        backend:
          serviceName: todo-webui-service
          servicePort: 80
      - path: /api/todos
        backend:
          serviceName: todo-api
          servicePort: 8080
      - path: /api/user
        backend:
          serviceName: user-api
          servicePort: 8082
  #The service that will handle all other
  #paths. e.g. return a 404 not found page.
  backend:
    serviceName: default-http-backend
    servicePort: 80
```



ConfigMaps and Secrets

- 12 Factor App #3
  - “Store config in the environment”
- Two resources (very similar):
  - ConfigMap
  - Secrets (base64 encoded)
- The config/secret can be exposed as:
  - Environment variables (use going forward)
  - Volumes (used for legacy apps e.g. existing spring application.properties)

DaemonSets

- Often you will want to schedule one pod on every node
- Such a pod is often called a system pod because they perform system specific duties
- For example,
  - Log aggregation
  - Monitoring
  - Security Event Monitoring
  - Etc
- See k8s-bootcamp/daemonsets

Jobs

- Jobs are those processes that run to completion
- They can be scheduled (with CronJobs)
- They can be parallelized
- They either complete successfully or not
- They can be configured to ”try again” up to XX times
- See k8s-bootcamp/jobs

StatefulSets

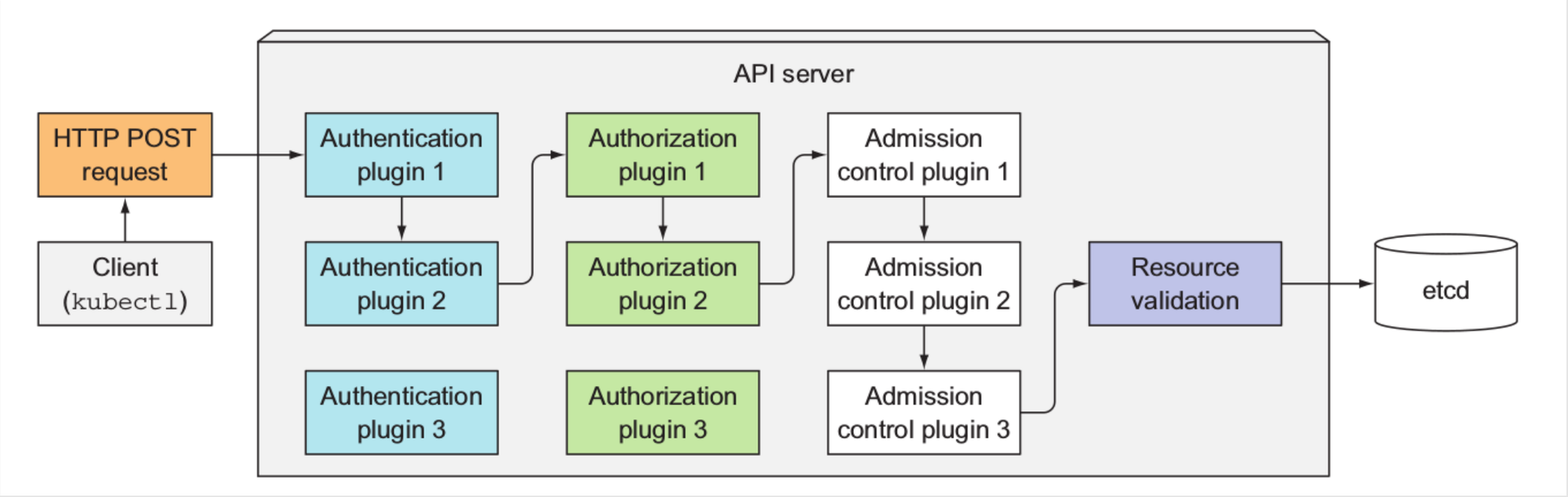
- Similar to deployments in that you can specify replicas and can do controlled upgrades and rollbacks
- Different in that each pod has identity and the PersistentVolumeClaims have identity
- Example:
  - MySQL cluster has a master node and replicated slaves
  - The master is read/write, the slaves are read-only
- See k8s-bootcamp/statefulsets

Securing Your Cluster

- There are many considerations that need to be made to secure you cluster. I will be talking about:
  - Authn/Authr
  - Pod Security Policies
  - Network Policies
- You also need to deal with rolling over keys, secret management, auditing, quickly revoking access, security event monitoring, encryption at rest, process, and more…
- See k8s-bootcamp/security

Authn/Authr

- Authr decision based on a set of attributes:
  - apigroup, resource, verb, subject
- Modules executed in order, short-circuit upon successful authentication/authorization



From "Kubernetes in Action" by Manning

RBAC

- Roles define permissions
- RoleBindings grant permissions to subject at a namespace level

```yaml
kind: Role
apiVersion: rbac.authorization.k8s.io/v1 #for v1.8.0++
#apiVersion: rbac.authorization.k8s.io/v1beta1 #for v1.7.x
metadata:
  name: production-role
  namespace: production
rules:
- verbs:
  - "create"
  - "delete"
  - "update"
  - "patch"
  - "get"
  - "list"
  - "watch"
  apiGroups: ["*"]
  resources: ["*"]
```

```yaml
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1 #for v1.8.0++
#apiVersion: rbac.authorization.k8s.io/v1beta1 #for v1.7.x
metadata:
  name: production-deployer
  namespace: production
roleRef:
  kind: Role
  apiGroup: rbac.authorization.k8s.io
  name: production-role
subjects:
- kind: User
  name: jenkins
  namespace: production
  apiGroup: rbac.authorization.k8s.io
```

Advanced Scheduling

- The kube-scheduler is very smart, it general, it does what you need
- However, there are scenarios where you need more fine-grained control
- Example:
  - You want dedicated nodes for ElasticSearch (because they need fast disks and are resource-intensive)
  - You want dedicated nodes for you database cluster, as it simplifies backing up the data
- See k8s-bootcamp/scheduling

Monitoring and Log Aggregation

- There are many, many tools to perform monitoring, log-aggregation on K8S
- Monitoring is having visibility (real-time as possible) on resource usage, network traffic, etc on your cluster. You want to monitor, containers, pods, nodes, storage, network, latency, etc
- Log Aggregation is bringing all relevant logs (both system and application) together, indexing them so that that they are quickly searchable down to the finest level of granularity possible.

Monitoring and Log Aggregation

Some tools:

- Prometheus for scraping monitoring data, Grafana for visualization
- FileBeat/LogStash/Fluentd to aggregate/index logs to be searchable by ElasticSearch, Kibana for visualization
- Azure OMS with Log Analytics
- Sysdig
- DataDog
- Splunk
- Many more!!!

Extending Kubernetes

- K8S has been designed from the outset to be extendible
- Many extension points (See http://bit.ly/2pugymE)
- We will cover the following:
  - Custom Resource Definitions
  - Aggregated APIs
  - Service Catalog (Open Service Broker API)
- See k8s-bootcamp/extending-k8s

Service-Meshes

- Service meshes have been around for a few years and it is a rapidly moving space and important for large-scale micro-services deployments
- Problems that they solve:
  - Distributed tracing of requests
  - Fine grained security policy
  - More intelligent routing
  - Circuit breakers in the network, not in the code
- See k8s-bootcamp/service-mesh

Cluster Administration

- There are many considerations when it comes to administrating a cluster. I will cover:
  - CNI (Container Network Interface)
  - LimitRanges
  - ResourceQuotas
  - Upgrading the Cluster
  - Backing up your cluster
- See k8s-bootcamp/cluster-admin

Options for Deploying K8S On-Premise

- Kubernetes is very flexible, and there are many options to deploy Kubernetes on-premise
- Let’s cover various options and considerations
- See k8s-bootcamp/on-premise

Openshift

- Kubernetes distribution from Redhat
- Positioned as a “PaaS”
- Extends core Kubernetes with additional API primitives and controllers to provide Developer and Operations workflows
- Provides the ‘oc’ cli that extends ‘kubectl’ with Openshift specific commands

Openshift Versions

- Multiple versions:
  - Openshift Online – Redhat hosted (AWS), multi-tenant
  - Openshift Dedicated – Redhat hosted (AWS), single-tenant
  - Openshift Container Platform – Deploy anywhere, Redhat supported
  - Openshift Origin – Open-source, upstream project for all other Openshift versions, Redhat does not provide support
  - Managed Openshift on Azure – Redhat and Microsoft support currently in Private Preview (https://azure.microsoft.com/en-ca/blog/openshift-on-azure-the-easiest-fully-managed-openshift-in-the-cloud/)
- Redhat supported versions have to use RHEL or RHEL Atomic as the master/worker node OS.
  - Origin supports CentOS and Ubuntu (flaky)

Openshift API Resources

- Openshift adds many additional API resources to the core Kubernetes primitives
  - Some overlap and therefore, you need to make a decision which one to use
  - Others provide an abstraction over underlying K8S resources
  - Others are new concepts that do not have similar capabilities in K8S e.g. BuildConfig, ImageStream
- To see all the Openshift API resources see: https://docs.okd.io/latest/rest_api/index.html
- Openshift also introduces their own custom controllers to consume these API resources to reconcile the ”current state” with the “desired state”

Key Openshift Resources

- Project
  - essentially analogous to K8S namespaces
- BuildConfig
  - describes how inputs are transformed into deployable artefacts
  - Supports multiple ”strategies” such as S2I (source to image), Docker, Pipeline (Jenkins), Custom
- DeploymentConfig
  - Analogous to K8S Deployment resource (Note, Openshift uses ReplicationControllers vs the Deployment/Replicaset model of Kubernetes)
  - Specify replicas, update strategy, etc
  - Supports “triggers” to redeploy based on change to another resource

Key Openshift Resources

- ImageStream
  - Essentially a virtual pointer to Docker images
  - Supports referencing different versions of a docker image
  - Referenced within BuildConfig and DeploymentConfig as image repositories
  - Changing an ImageStream can trigger builds and deployments
- Route
  - Similar to K8S ingress rules, used to specify routing rules consumed by “Routers”
  - Default HAProxy router is deployed as a Pod on the cluster
  - Supports sharding, blue/green, canary, A/B testing deployment scenarios

Key Openshift Resources

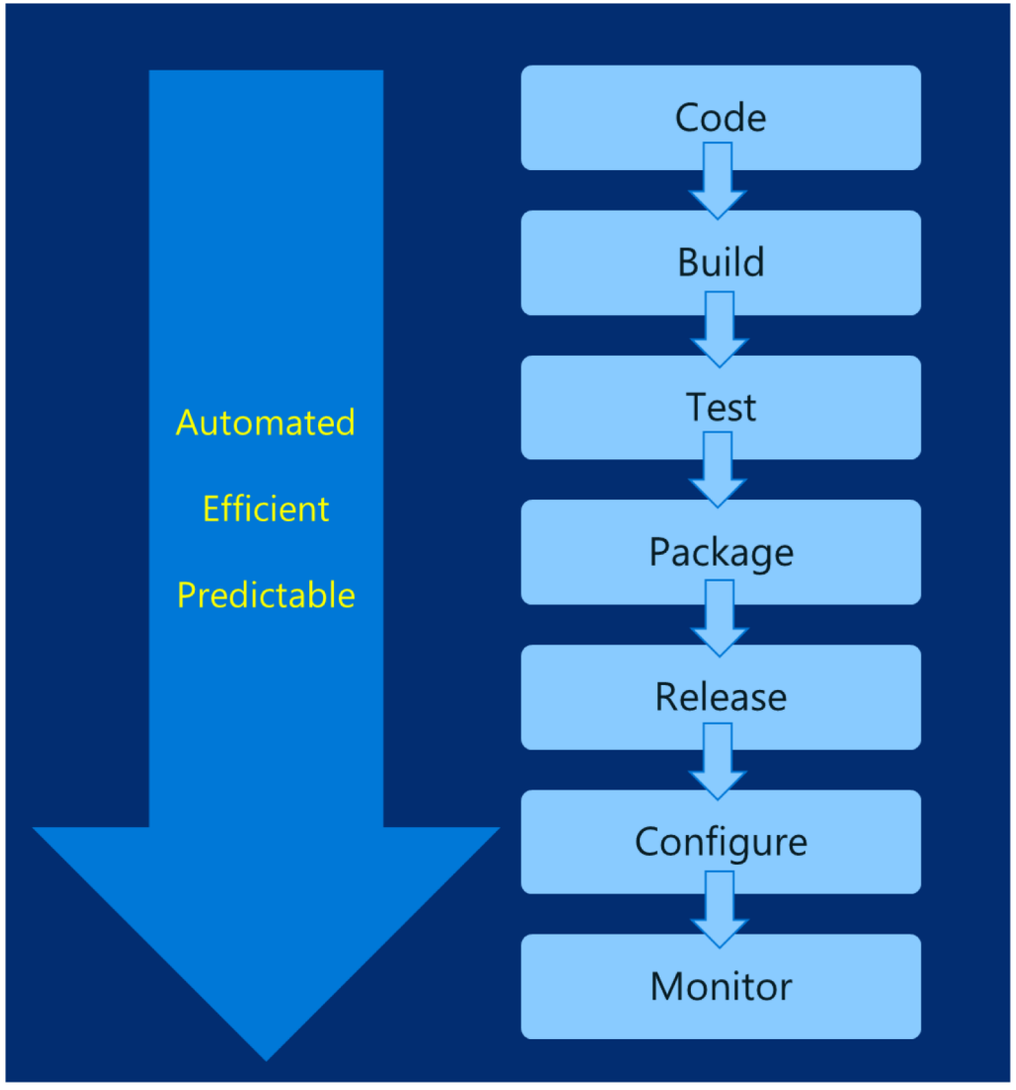
- SecurityContextConstraints
  - Similar to PodSecurityPolicies
- Template
  - A parameterized template to deploy multiple API resources that make up a solution
  - Used to specify service catalog components in the Web Console
  - Helm and helm charts are much more powerful
- User/Identity/Group
  - K8S does not have a notion of a User. K8S supports multiple identity stores and maps identity (via multiple approaches) to RBAC policy. Note, Openshift does use K8S RBAC model.

DevOps &
CI / CD

DevOps Practices Arrive

- Developers: Test, Build, Code, Plan
- Operations: Monitor, Release, Deploy, Operate
- DevOps Features
  - Speed of application delivery/updates
  - Faster time to value
  - Repeatable/consistent
  - Automated testing
  - Traceability of process
  - Applying developer patterns to infrastructure
    - Infrastructure as code
    - Ops teams embracing source control
  - Centralized monitoring, logging, debugging
- CI / CD – key aspect of quality DevOps

DevOps – Why you care



Deploy 200 times more frequently

Go from code check-in to production 2,555 times faster

Recover from failure 24 times faster

Spent 50% less time remediating security challenges

Spent 22% less time on unplanned work

2.2 times more likely to believe their places a great place to work

Source: https://puppet.com/resources/white-paper/2016-state-of-devops-report

Common Toolsets

- Jenkins
- Visual Studio Team Services
- Spinnaker and Netflix OSS
- Travis
- TeamCity
- CircleCI
- Additional utilities: code quality scanning, security, collaboration, etc.

Helm:
Kubernetes Package Manager

Birth of Helm

On October 15th, 2015

Hackathon project at company offsite

> Could we take the ideas behind npm and Homebrew and build something for deploying apps into Kubernetes?

Installation tool for Deis Workflow

Announced at the first KubeCon in San Francisco 2015

Helm Charts

Application definition

Consists of:

- Metadata
- Kubernetes resource definitions
- Configuration
- Documentation

Stored in chart repository

- Any HTTP server that can house YAML/tar files (Azure, GitHub pages, etc.)
- Public repo with community supported charts (eg – Jenkins, Mongo, etc.)

Helm (CLI) + Tiller (server side)

Release: Instance of chart + values -> Kubernetes

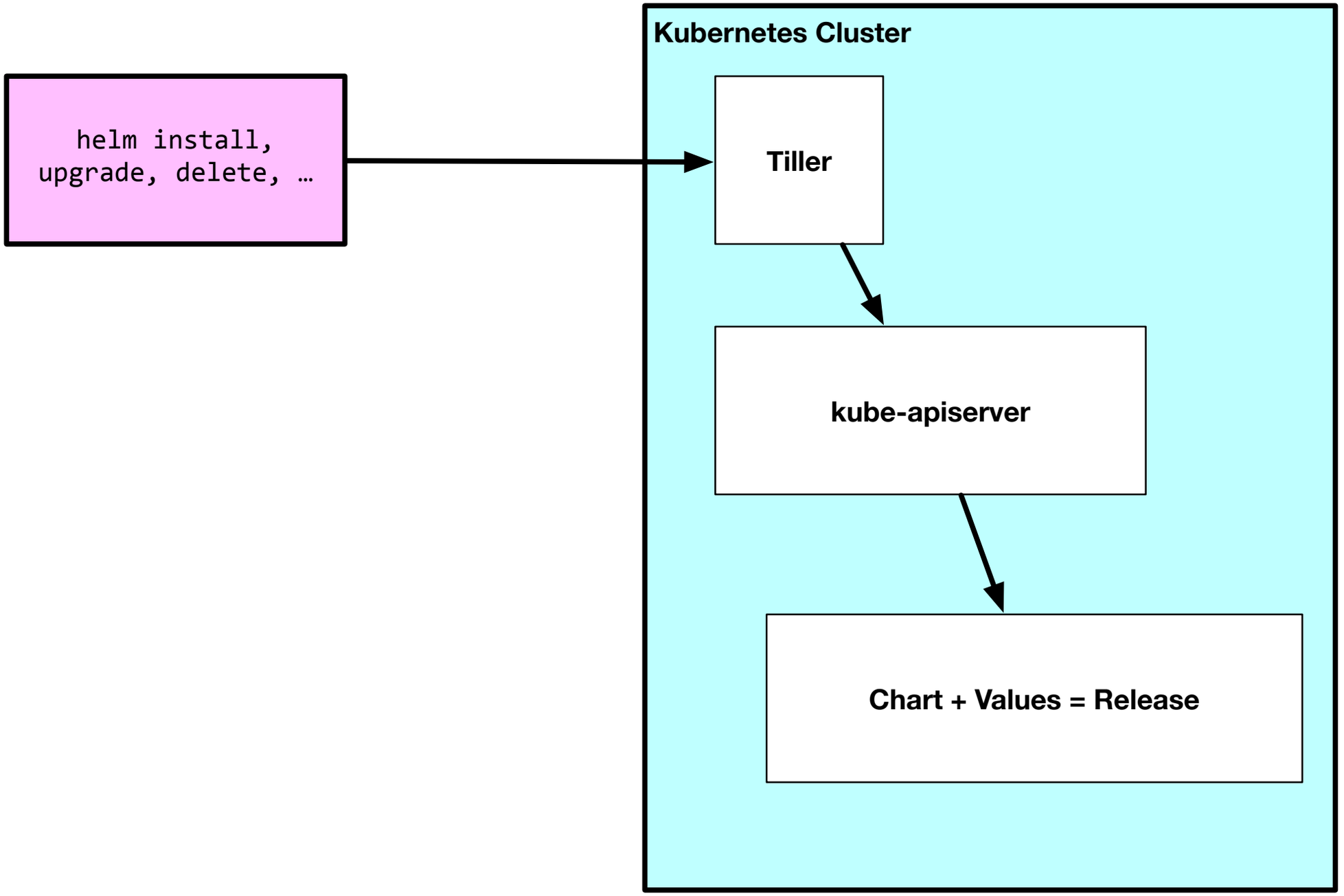

helm install,
upgrade, delete, …
Kubernetes Cluster
Tiller
kube-apiserver
Chart + Values = Release

Chart structure

- Charts have structure
  - Set of conventions, including file and directory names
  - Charts can be packaged into tarballs for distribution

Chart structure

- Layout
  - Helm expects a strict chart structure

```
wordpress/
  Chart.yaml          # A YAML file containing information about the chart
  LICENSE             # OPTIONAL: A plain text file containing the license for the chart
  README.md           # OPTIONAL: A human-readable README file
  values.yaml         # The default configuration values for this chart
  charts/             # OPTIONAL: A directory containing any charts upon which this chart depends.
  templates/          # OPTIONAL: A directory of templates that, when combined with values,
                      # will generate valid Kubernetes manifest files.
  templates/NOTES.txt # OPTIONAL: A plain text file containing short usage notes
```

Chart.yaml

```
name: The name of the chart (required)
version: A SemVer 2 version (required)
description: A single-sentence description of this project (optional)
keywords:
  - A list of keywords about this project (optional)
home: The URL of this project's home page (optional)
sources:
  - A list of URLs to source code for this project (optional)
maintainers: # (optional)
  - name: The maintainer's name (required for each maintainer)
    email: The maintainer's email (optional for each maintainer)
engine: gotpl # The name of the template engine (optional, defaults to gotpl)
icon: A URL to an SVG or PNG image to be used as an icon (optional).
```

Helm values.yaml

- The knobs and dials:
  - A `values.yaml` file provided with the chart that contains **default** values
  - Use -f to provide your own values overrides
  - Use --set to override individual values

Helm Templates

- Built on Go's template language w/addition of 50 or so add-on template functions
- Almost anything goes! ;-)
- Also useful in generating random values (e.g. passwords)
  - Provides flow control (if/else, with, range, etc)
  - Named templates (partials)

Open Service Broker for Azure

- Connect Kubernetes apps to Azure simply and securely
  - Azure Database for MySQL
  - Azure Database for PostgreSQL
  - Azure SQL
  - Azure CosmosDB
  - Azure Redis
  - Azure Container Instances
  - Azure Service Bus
  - Azure Storage
  - More to come…
- Built on an open standard
- Integrated with Helm

Draft: Streamlined Kubernetes Development

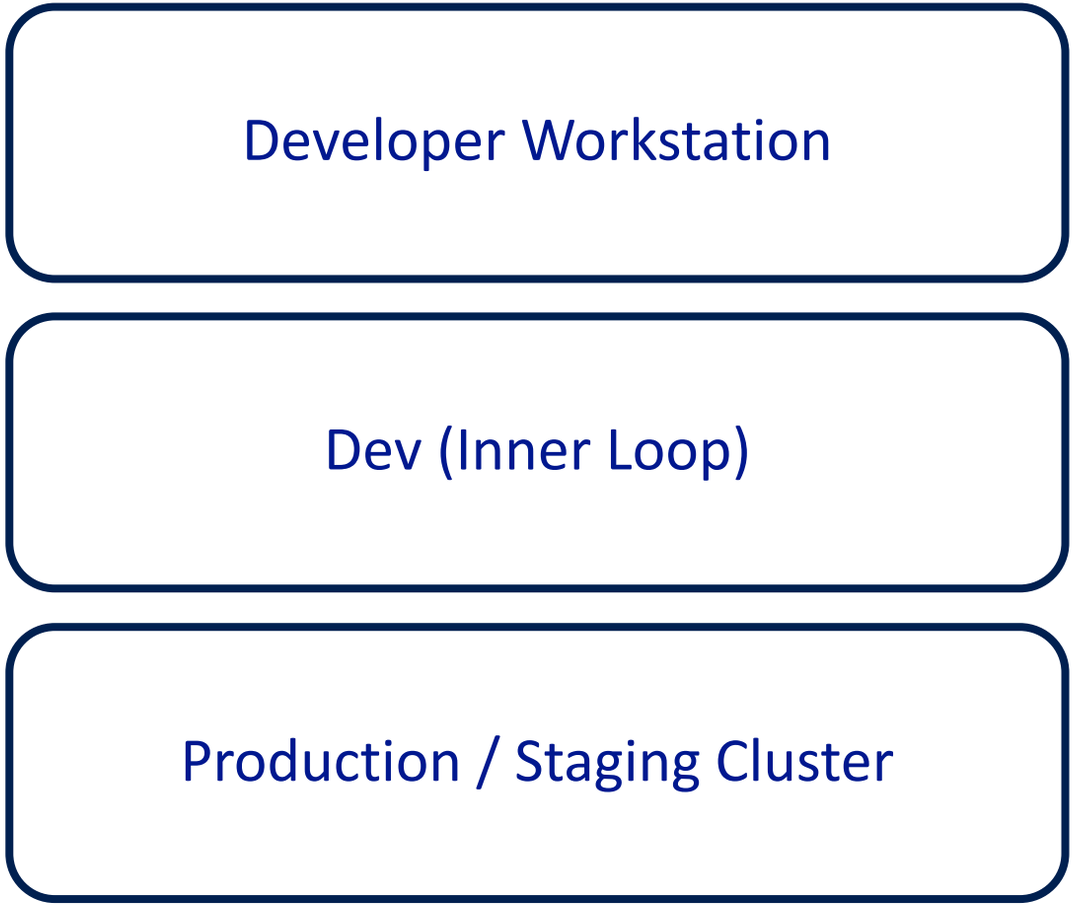
What is draft?

Developer workstations cannot mimic production

As devs hack on code prior to committing to version control

Draft targets the "inner loop" of a developer's workflow

Works with other Deis tools such as Helm and Workflow



DRAFT

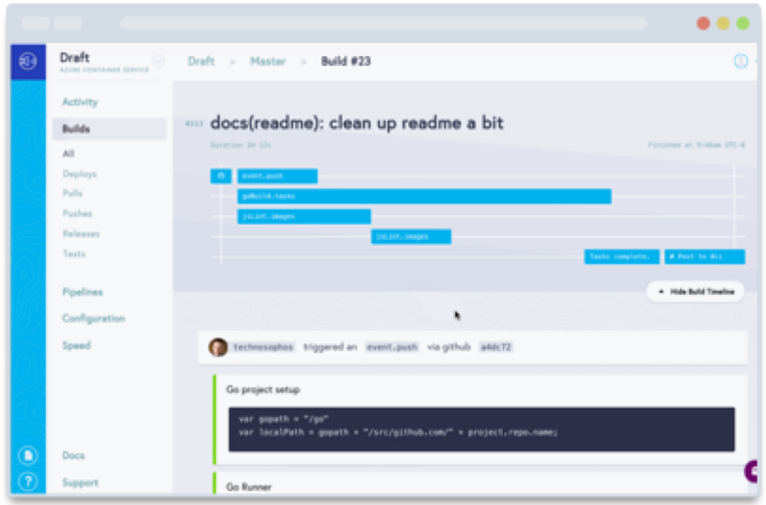
brigade: event-driven scripting for kubernetes

- encapsulate functions in containers
- run in parallel or serial
- trigger workflow from Github, Docker registry, etc.
- javascript (pipeline as code)
- project config stored as secrets
- well suited for CI/CD pipelines

kashti: a dashboard for brigade projects

Kashti

- easy viewing and constructing brigade pipelines
- waterfall diagram of brigade builds
- view log output of build steps
- quick peek at configuration for builds



Questions?

WHO WE ARE

Design. Build. Transform.

www.architech.ca
kubernetes@architech.ca

As a digital product studio, we enable digital transformation up to 87% faster.

13+ years

100+ people

300+ projects

TELUS the future is friendly® · TD · COMPASS GROUP · EllisDon We build on great relationships™ · IRON MOUNTAIN® · Ontario · SPORTCHEK · ARCHITECH

LEGACY MODERNIZATION

Migrate and rewrite applications to modern cloud architectures

DEDICATED TEAMS

Accelerate the delivery timelines of open source development projects

PRODUCT INNOVATION

Lean product innovation from concept to MVP

DIGITAL PRODUCT STUDIO

Teams	Mastery Excellence	Integrated Collaborative	Skilled Certified	Agile Lean
Capabilities	Strategy + Design + Engineering + Cognitive Analytics			
Methodology	Lean Startup + Design Thinking + Agile Engineering			
Technology	Industry Proven | Lightweight | Open Source | Cloud-First			
Partners	Microsoft	redhat.	dotCMS	scalar

The right, quality solution, fast.

TECHNOLOGIES & PARTNERS

Application	
Cognitive Analytics	
DevOps CI/CD	
Infrastructure	
Platform	
Operations & Support