

Welcome

Intermediate Cloud Computing

Hello



About me...



- 25+ years in the industry
- 20+ years in teaching
- Certified Cloud architect
- Passionate about learning
- Also, passionate about Reese's Cups!

Prerequisites

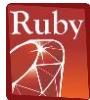
This course assumes you:

- Have foundational experience in one or more modern public Cloud Service Providers (CSPs) – e.g., AWS, Azure, or Google Cloud

Why study this subject?

- Cloud is everywhere and it's not going away
- As with many topics in technology, there are multiple options and multiple dimensions to those options
- Building a deeper understanding of Cloud and its offerings positions you to help your customers with their Digital Transformation journey

We teach over 400 technology topics



You experience our impact on a daily basis!



My pledge to you

I will...

- Make this interactive
- Ask you questions
- Ensure everyone can speak
- Use an on-screen timer

Objectives

At the end of this course you will be able to:

- Speak to key architectural and design considerations relevant for effective Cloud development
- Understand, at a high-level, how Infrastructure-as-Code (IaC) enables Cloud development
- Understand, at a high-level, how containerization enables Cloud development

Agenda

- Infrastructure and hosting options with Cloud
- Key design principles and best practices to guide Cloud builds
- Planning, deploying, and managing Cloud services
- Infrastructure-as-Code (IaC) using Terraform
- Containerization in the Cloud using Docker and Kubernetes

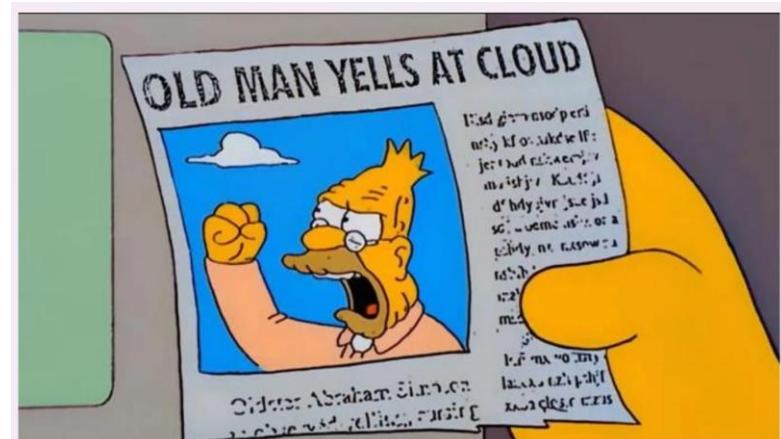
How we're going to work together

- Lecture & slides
- Hands-on labs
- Class discussions
- Borathon

Open Discussion

What is “the Cloud”?

How does “the Cloud” impact (or even enhance) our approach to business & technology enablement?



Infrastructure Options

What Are the Options?

- On-Premise
- Public Cloud
- At the Edge
- Hybrid Cloud

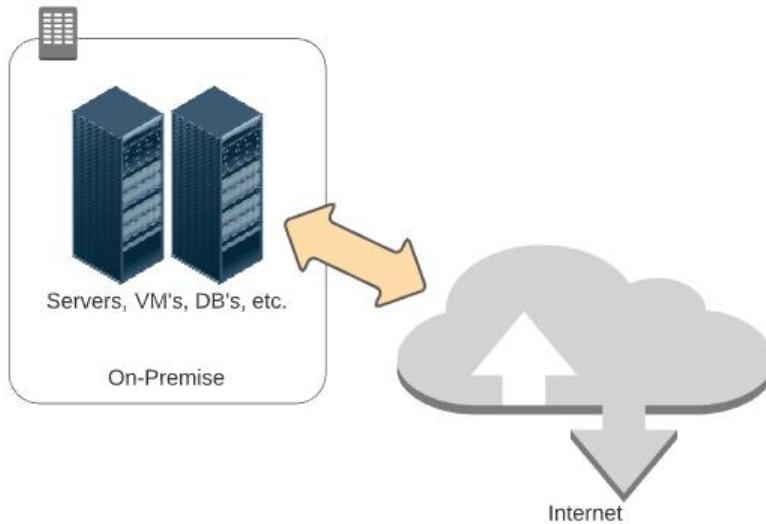
What do they all mean?

On-Premise

Can mean a few different things:

- In a wholly-owned Data Center
- In a COLO (or co-location Data Center)
- Sometimes called a “private cloud”

On-Premise



On-Premise

Why and What?

- It's how infrastructure has traditionally been done
- With this model, companies try and estimate hardware capacity needed to support business operations
- Stakeholders look to plan out expected levels of consumption for the next 3 – 5 years (capacity to handle current volumes as well as expected growth)
- Some critical workloads may not be suitable for anything but a physical and directly-managed implementation (e.g., mainframe)

On-Premise – Discussion

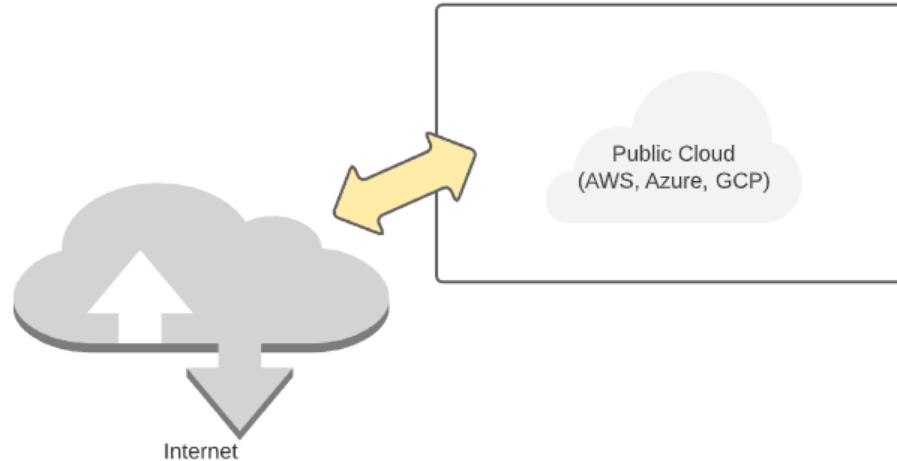
Pros?

- Discrete capacity planning (even if that planning was off)
- Some workloads (e.g., mainframes and certain legacy systems) are tailor-made for a physical data center
- With a move to COLO's, companies could begin to share expenditure

Cons?

- Sometimes difficult to know what is needed and when it is needed – if the plan was off (or unexpected spikes in demand occurred), difficult to adjust quickly
- Some workloads are just as effective (if not more so) in a virtual vs. physical implementation
- Harder to control costs and plan for costs – CAPEX vs. OPEX

Public Cloud



Public Cloud

Why and What?

- Platform using the standard “Cloud computing model” to provide infrastructure and application services
- Accessed and integrated via the Internet
- May provide a few different types of services – IaaS, PaaS, etc.
- Usually supports a subscription or “pay as you go” (on-demand) pricing model
- Largest players in this space include Azure, AWS and GCP

Public Cloud - Discussion

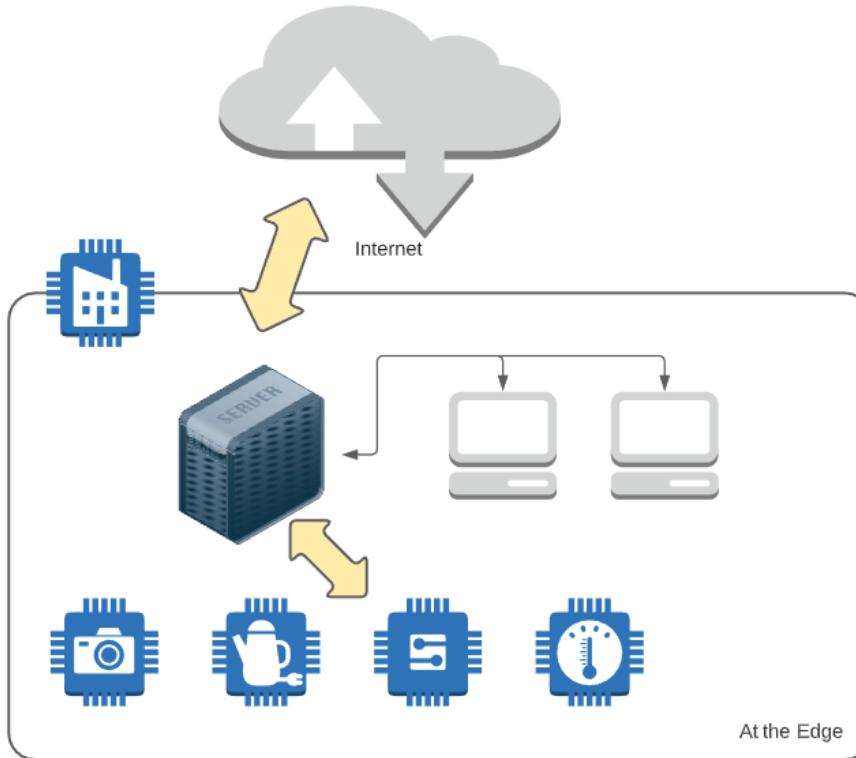
Pros?

- Flexibility and elasticity in capacity planning
 - enables automated schedule-based or metrics-based adjustments to capacity when required
- In some cases, managed services can be leveraged reducing operations overhead
- Because services are PAYG (pay as you go), you're only charged for what you use, and those expenses are OPEX

Cons?

- Requires enough historical data for schedule-based planning or the right configuration for metrics-based planning
- With managed services you lose some levels of granular control
- Because of the flexibility/elasticity, it can be difficult to budget and, if Cloud services are not managed/monitored, costs can be high

At the Edge



At the Edge

Why and What?

- It's about bringing the power of Cloud computing to you
- Enables additional processing closer to the sources of data while still supporting the offload of higher order processing to the Cloud
- Often involves setting up “Cloud-in-a-box” facilities on-premise
- IoT (Internet of Things) is a good example – devices in a facility reading massive amounts of data can incorporate processing at the edge to improve overall efficiency
- Helps inject lower latency, increased security and improved bandwidth into systems used to aggregate critical data for an enterprise

At the Edge - Discussion

Pros?

- Allows distribution of processing power across a larger surface area
- Can be used to bring critical latency, security and bandwidth improvements to specific types of business workflows
- Efficiencies gained “at the edge” can help with managing the cost of processing data

Cons?

- Requires more infrastructure and more configuration to support that distribution
- Increased distribution of processing power and activity can expand attack surface and requires the right configuration to ensure optimal interaction between system components (i.e., increased complexity)
- More components “at the edge” can lead to increased infrastructure costs

Hybrid Cloud

Why and What?

- In many ways, an amalgamation of the other options
- Supports distribution of system processing across on-premise infrastructure and the public Cloud
- Allows an enterprise to keep workloads that are best-suited for on-premise running on-premise while allowing migration of components that can move to the public Cloud
- Can help make an enterprise's move to the Cloud more gradual and planful

Hybrid Cloud - Discussion

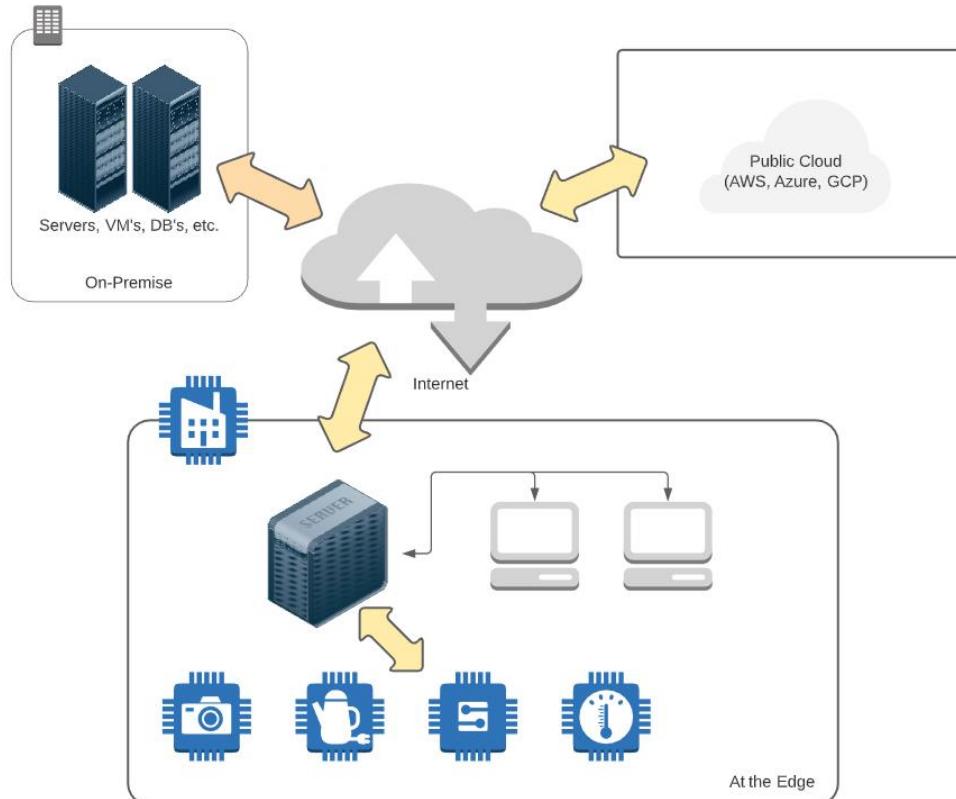
Pros?

- Allows distribution of processing power across a larger surface area
- Can allow a move to the Cloud to be more gradual and allow an enterprise to target optimal deployment platform while making the move
- The ability to support a gradual move enables an enterprise to assess and understand Cloud costs over time

Cons?

- Requires more infrastructure and more configuration to support that distribution
- As with Edge, can lead to increased complexity, often including required setup and maintenance of dedicated, secure connectivity between a data center and the Cloud
- If not managed optimally, costs can be higher due to need to pay for Cloud usage and data center (CAPEX + OPEX)

Hybrid Cloud



Lab – Getting Setup

Application Hosting

What Do We Mean by Hosting?

- The target infrastructure and runtime platform that will be employed for deployment and execution of an application or system
- Can include compute (CPU and server resources), storage, network, data and operating system

Application Hosting – An “Interesting” Example?

Here's an example of someone thinking “outside-of-the-box” when it comes to application hosting!

<https://mashable.com/article/pregnancy-test-doom/>

What Are the Hosting Options with Cloud?

- IaaS
- PaaS
- Serverless / FaaS
- SaaS
- Containers (we'll look at that topic separately)

What do they all mean?

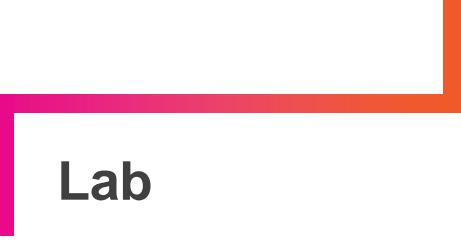
Infrastructure-as-a-Service (IaaS)

- Involves the building out (and management) of virtual instances of:
 - Compute
 - Network
 - Storage
- Akin to spinning up a server (physical or virtual) in your location or data center complete with disks and required network connectivity
- The difference is in the where – instead of in your data center, it is created in a data center managed by one of the public Cloud providers
- Your organization is responsible for patching the OS, ensuring all appropriate security updates are applied and that the right controls are in place to govern interaction between this set of components and other infrastructure

Platform-as-a-Service (PaaS)

- Involves leveraging managed services from a public Cloud provider
- With this model, an enterprise can focus on management of their application and data vs. focusing on management of the underlying infrastructure
- Patching and security of the infrastructure used to back the managed services falls to the CSP (Cloud Service Provider)
- Many managed services support automatic scale up or down depending on demand to help ensure sufficient capacity is in place
- Part of what is often termed the “Shared Responsibility Model”

Demo



Lab

Serverless / Functions-as-a-Service (FaaS)

- Also represents a type of managed service provided by the CSP
- Cost structure is usually consumption-based (i.e. you only pay for what you use)
- Supports many different coding paradigms (C#/.NET, NodeJS, Python, etc.)
- Typically, with Serverless (and PaaS), the consumer is only concerned with the application code and data – elements of the CSP's "backbone" used to support are managed by the CSP
- Includes more sophisticated automated scaling capabilities – built for Internet scale

AWS Lambda Using Go

- Lambdas can be created in the AWS Management Console
- Allows language selection and testing or can use a blueprint as a starting point
- The in-browser IDE does not support Go which prevents editing in the Management Console

AWS Lambda Using Go

- But we're able to zip it up, push it to an S3 bucket, and then deploy from there
- Likely a better experience than developing than the MC anyway
- In addition to the Lambda, you'll want an API Gateway in place as well

AWS Lambda Using Go

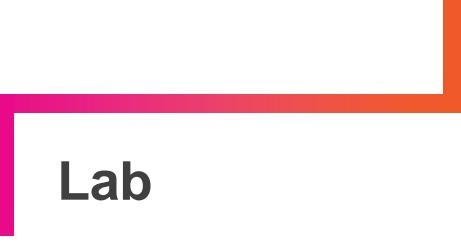
- This Gateway will provide an HTTP REST interface to the Lambda's operations
- The Gateway will be setup as a trigger (it's one of Lambda's standard triggers)
- The Gateway includes internal components that can proxy to/from the Lambda

AWS Lambda Using Go

- Go packages have been provided by AWS for coding the Lambda's handler
- Provide several utility functions for managing function execution and results processing

Demo

<https://github.com/ludesdeveloper/terraform-lambda-golang>



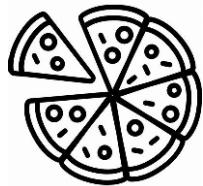
Lab

Software-as-a-Service (SaaS)

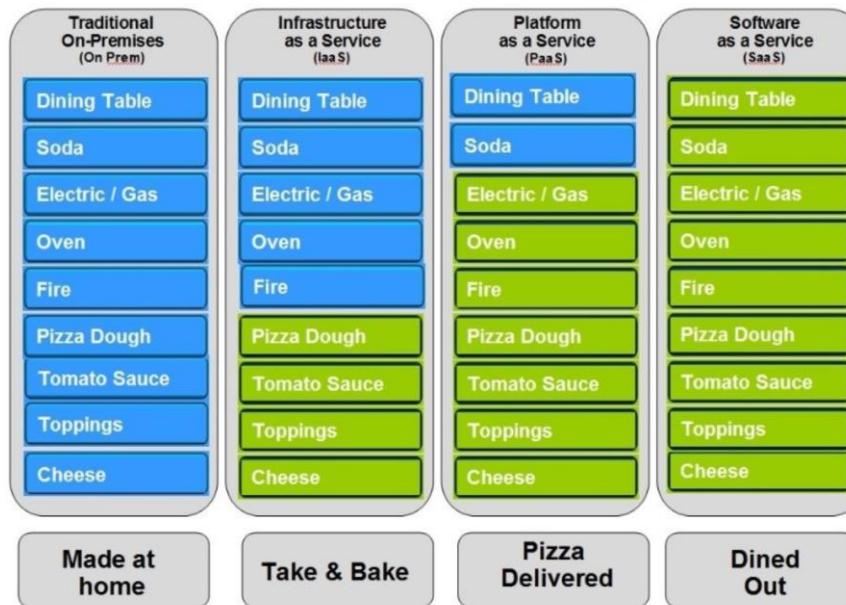
- Subscription-based application services
- Licensed for utilization over the Internet / online rather than for download and install on a server or client machine
- Fully-hosted and fully-managed by a 3rd party
- Of those discussed, often the cheapest option for service consumers
- However, also offers minimal (or no) control, outside of exposed configuration capabilities

Pizza-as-a-Service

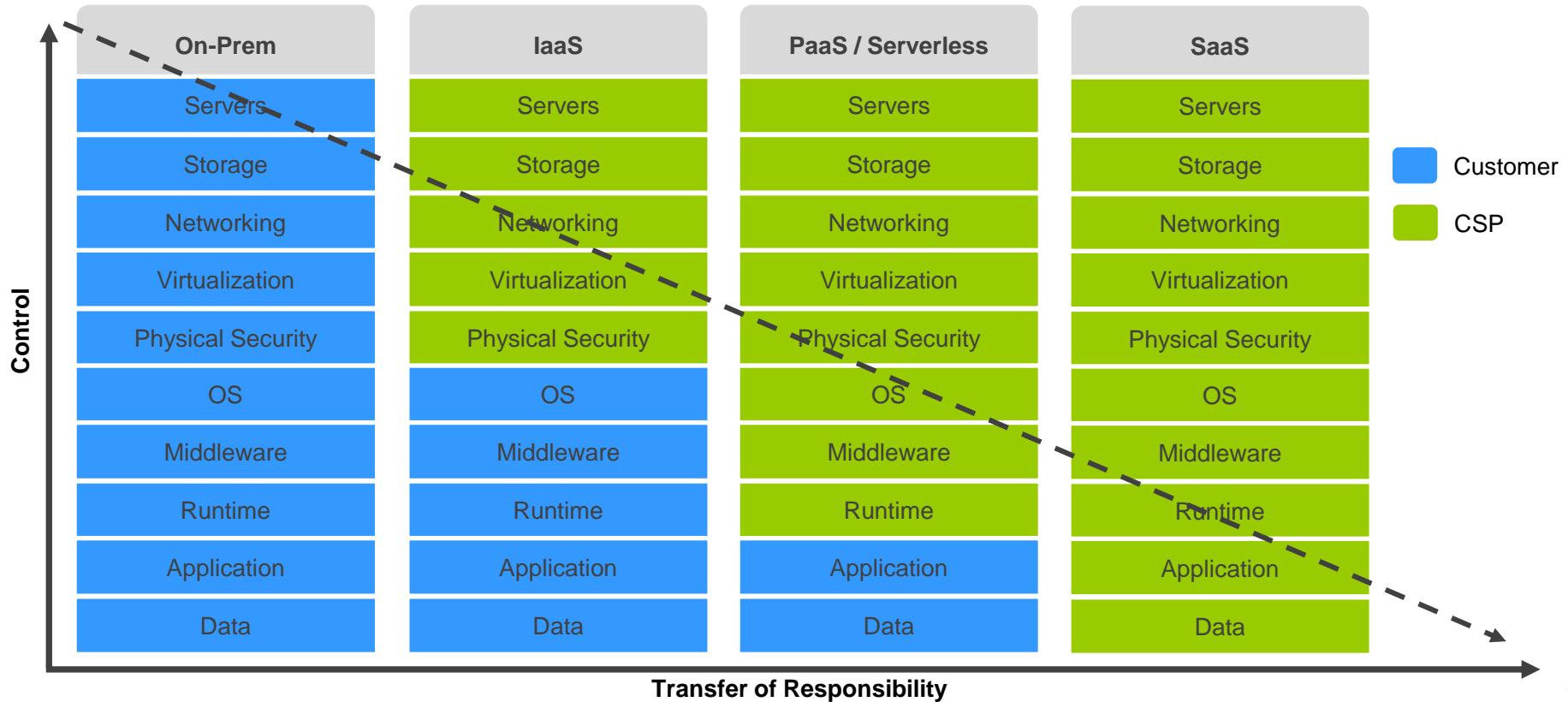
From a LinkedIn post by Albert Barron from IBM (<https://www.linkedin.com/pulse/20140730172610-9679881-pizza-as-a-service/>)



Pizza as a Service



Side-by-Side Comparison



Which One is Better?

- The answer is “it depends”
- It depends on the type of application
- It depends on the enterprise
- It depends on the skillset and expertise within the organization
- It depends on whether you have budget and opportunity to modernize an application environment (in some cases)
- The best option might be a combination of multiple approaches – right tool for the right job

Case Study – Discussion

Scenario: A global supplier is looking to modernize its order processing system. Among other things, this system includes Accounts Payable, Accounts Receivable, Inventory, Shipping, Invoicing, and Raw Materials modules. The majority of their current system is built on a mainframe stack which is several versions behind in terms of system upgrades and patches, and, while the modules used by the company are provided with out-of-the-box implementations by the mainframe vendor, the company has chosen to heavily customize those modules over multiple years. Internal users interface with the modules of the system using “green screen” terminals. There is a thin web layer for presentation to and utilization by external users which submits Message Queueing (MQ) calls to the mainframe for the back-end business logic.

Let's discuss:

- What kinds of requirements gathering questions would you ask this customer as you begin helping them plan for modernization?
- What kinds of infrastructure and hosting recommendations would you provide to this customer for technical implementation of a modernized solution in the Cloud?



SOLID Principles & Good Design

Architecting for the Future

- When we architect and build an application at a “point in time”, we hope that the application will continue to be utilized to provide the value for which it was originally built
- Since the “only constant is change” (a quote attributed to Heraclitus of Ephesus), we have to expect that the environment in which our application “lives and works” will be dynamic
- Change can come in the form of business change (change to business process), the need to accommodate innovation and ongoing advancement in technology
- Often, the speed at which we can respond to these changes is the difference between success and failure

Architecting for the Future

In order to ensure that we can respond to change “at the speed of business”, we need to build our systems according to best practices and good design principles:

- Business-aligned design
- Separation of concerns
- Loose coupling
- Designing for testability

Business-Aligned Design

- Build systems that use models and constructs that mirror the business entities and processes that the system is intended to serve
- Drive the design of the system and the language used to describe the system based around the business process not the technology
- AKA Domain Driven Design
- Results in a system built out of the coordination and interaction of key elements of the business process – helps to ensure that the system correlates to business value
- Also helps business and technology stakeholders keep the business problem at the forefront

Separation of Concerns

- Break a large, complex problem up into smaller pieces
- Drive out overlap between those pieces (modules) to keep them focused on a specific part of the business problem and minimize the repeat of logic
- Logic that is repeated, and that might change, will have to be changed in multiple places (error prone)
- Promotes high cohesion and low coupling (which we will talk about in a minute)
- Solving the problem becomes an exercise in “wiring up” the modules for end-to-end functionality and leaves you with a set of potentially reusable libraries

Loose Coupling

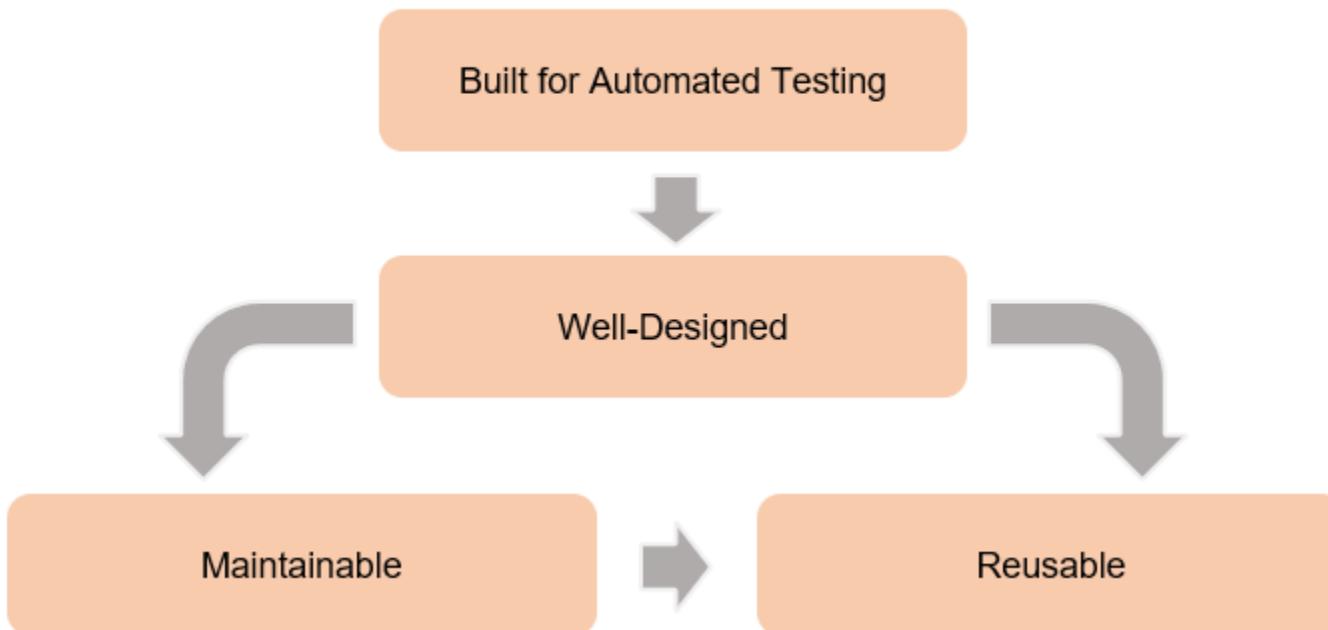
- Coupling between components or modules in a system causes problems, especially for maintaining the system over time
- System components that are tightly coupled, are more difficult to change (or enhance) – changes to one part of the system may break one or more other areas
- With coupling, you now must manage the connected components as a unit instead of having the option to manage the components in different ways (e.g. production scalability)
- Makes the job of unit testing the components more difficult because tests must now account for a broader set of logic and dependencies (e.g. tight coupling to a database makes it difficult to mock)

Designing for Testability

- Practicing the previous principles helps lead to a system that is testable
- Testability is important because it is a key enabler for verifying the quality of the system – at multiple stages along the Software Development Lifecycle (SDLC)
- When building a system, quality issues become more expensive to correct the later they are discovered in the development lifecycle – good architecture practices help you test early and often
- Ideally, testing at each stage will be automated as much as possible in support of quickly running the tests as and when needed

Designing for Testability

Testable code is...



SOLID Principles

SOLID principles help us build testable code

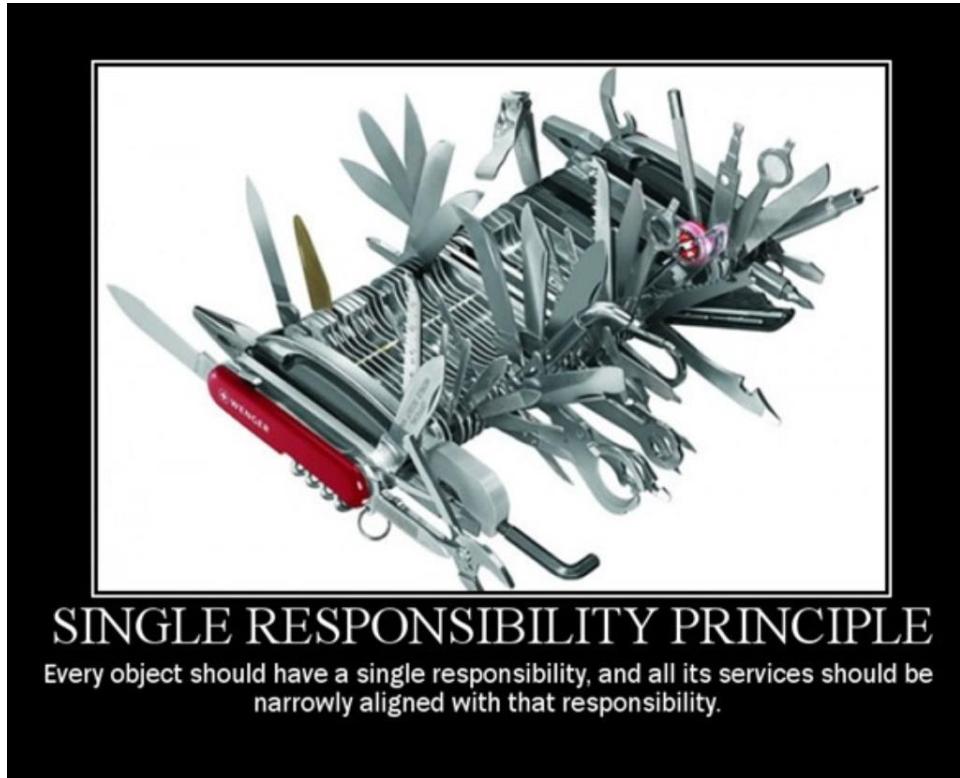
- Single Responsibility Principle (SRP)
- Open-Closed Principle (OCP)
- Liskov Substitution Principle (LSP)
- Interface Segregation Principle (ISP)
- Dependency Inversion Principle (DIP)

Single Responsibility Principle (SRP)

Single Responsibility Principle (SRP)

A system module or component should have only one reason to change

Single Responsibility Principle (SRP)



Single Responsibility Principle (SRP)

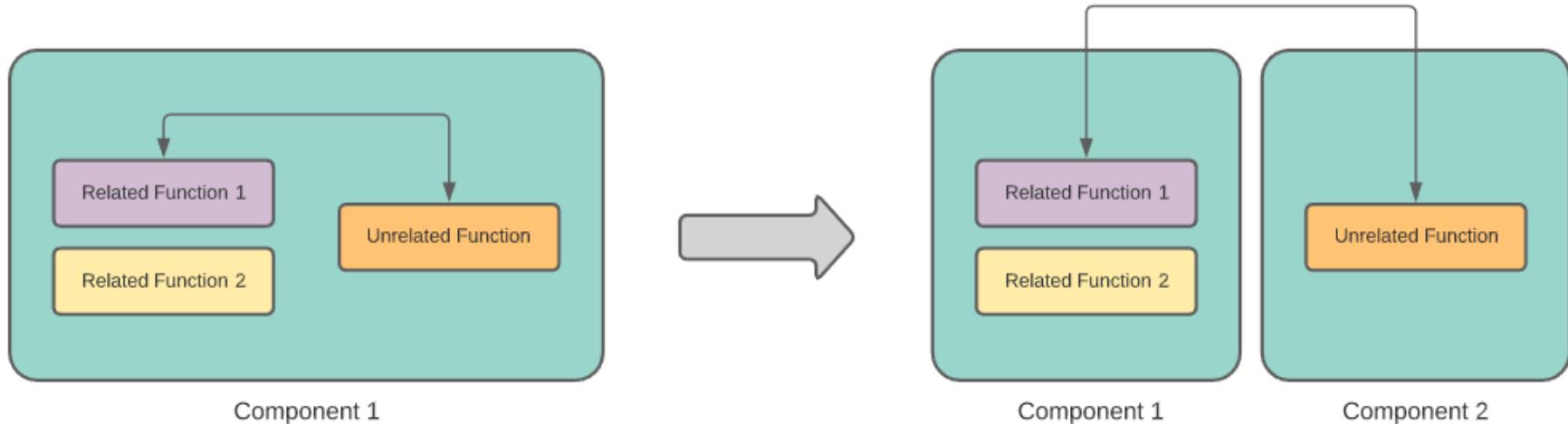
Why important?

- Related functions organized together breed understanding (logical groupings) – think cohesion
- Multiple, unrelated functionalities slammed together breed coupling
- Reduced complexity (cleaner, more organized code)
- Promotes smaller code modules
- Reduced regression – tension of change

Single Responsibility Principle (SRP)

```
5  namespace CoolCompany.Marketing
6  {
7      public class MarketingCampaign
8      {
9          #region Private Members
10
11         private readonly string[] addresses = new string[]
12         {
13             "customer.one@companya.com",
14             "customer.two@companyb.com",
15             "customer.three@companyc.com"
16         };
17
18         #endregion
19
20         #region Properties
21
22         public string Name { get; set; }
23         public string Description { get; set; }
24         public string ManagerName { get; set; }
25         public TimeSpan CampaignLength { get; set; }
26
27         #endregion
28
29         #region Public Methods
30
31         public void LaunchCampaign(TimeSpan campaignLength)
32         {
33             CampaignLength = campaignLength;
34             // Logic responsible for launching the new marketing campaign
35         }
36
37         public void SendEmails()
38         {
39             // Logic responsible for sending e-mails to list of addresses associated to campaign
40         }
41
42         #endregion
43     }
44 }
```

Single Responsibility Principle (SRP)



Single Responsibility Principle (SRP)

How to practice?

- When building new modules (or refactoring existing), think in terms of logical groupings
- Look for “axes of change” as points of separation
- Build new modules to take on new entity or service definitions – provides abstraction
- Structure clean integrations between separated modules
- Use existing tests (or build new ones) to verify a successful separation

Open-Closed Principle (OCP)

Open-Closed Principle (OCP)

Software entities should be open for extension but closed for modification

Open-Closed Principle (OCP)



OPEN CLOSED PRINCIPLE

Brain surgery is not necessary when putting on a hat.

Open-Closed Principle (OCP)

Why important?

- Our systems need to be able to evolve
- We need to be able to minimize the impact of that evolution

Open-Closed Principle (OCP)

How to practice?

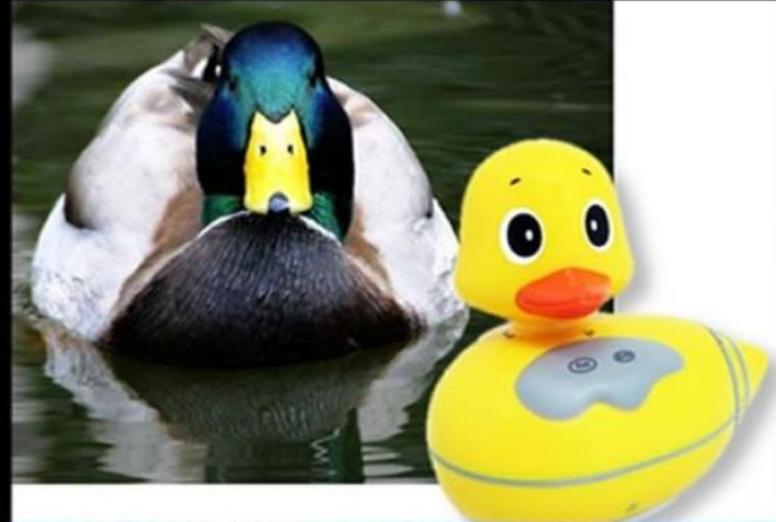
- When building new modules (or refactoring existing), leverage abstractions
- Use the abstractions as levers of extension
- Use existing tests (or build new ones) to verify the abstractions

Liskov Substitution Principle (LSP)

Liskov Substitution Principle (LSP)

Subtypes must be substitutable for their base types

Liskov Substitution Principle (LSP)



Liskov Substitution Principle

If it looks like a duck and quacks like a duck but needs batteries,
you probably have the wrong abstraction.

Liskov Substitution Principle (LSP)

Why important?

- Want to be able to use the abstractions created by OCP to extend existing functionality (vs. modify it)
- Especially useful when multiple variants of a type need to be processed as a single group
- Promotes looser coupling between our modules
- Without it, we may have to include if/else or switch blocks to route our logic
- Or keep adding parameters/properties for new but related types

Liskov Substitution Principle (LSP)

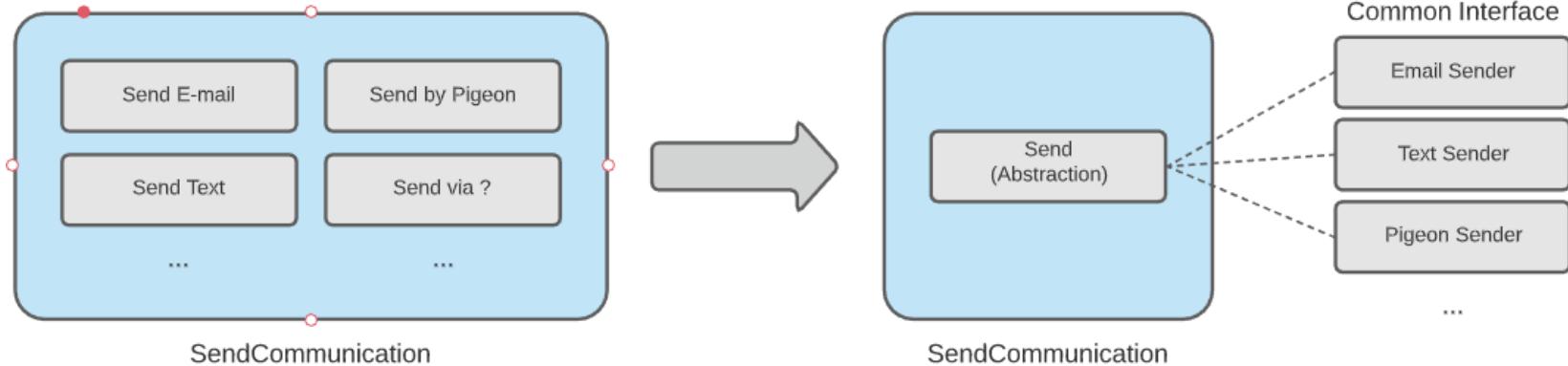
How to practice?

- Module members should reference the abstractions in consuming code
- Look for ways to encapsulate type-specific (or type-aware) logic in the type instead of in the code using the type
- Use existing tests (or build new ones) to verify our ability to effectively substitute

OCP & LSP

```
14      0 references
15  ┌─ public class CommunicationAgent
16  │
17  │  0 references
18  │  ┌─ public void SendCommunication(CommunicationType communicationType)
19  │  │
20  │  │  if (communicationType == CommunicationType.Email)
21  │  │  {
22  │  │      // Call to another component that knows about sending e-mail
23  │  │  }
24  │  │  else if (communicationType == CommunicationType.Text)
25  │  │  {
26  │  │      // Call to another component that knows about sending texts
27  │  │  }
28  │  │  else if (communicationType == CommunicationType.Pigeon)
29  │  │  {
30  │  │      // Call to another component that knows about sending messages by pigeon
31  │  │  }
32  │  │  else
33  │  │  {
34  │  │      // Error or default behavior - unrecognized communication type
35  │  │  }
36  │  │
37  │  ┘
38  ┘
```

OCP & LSP

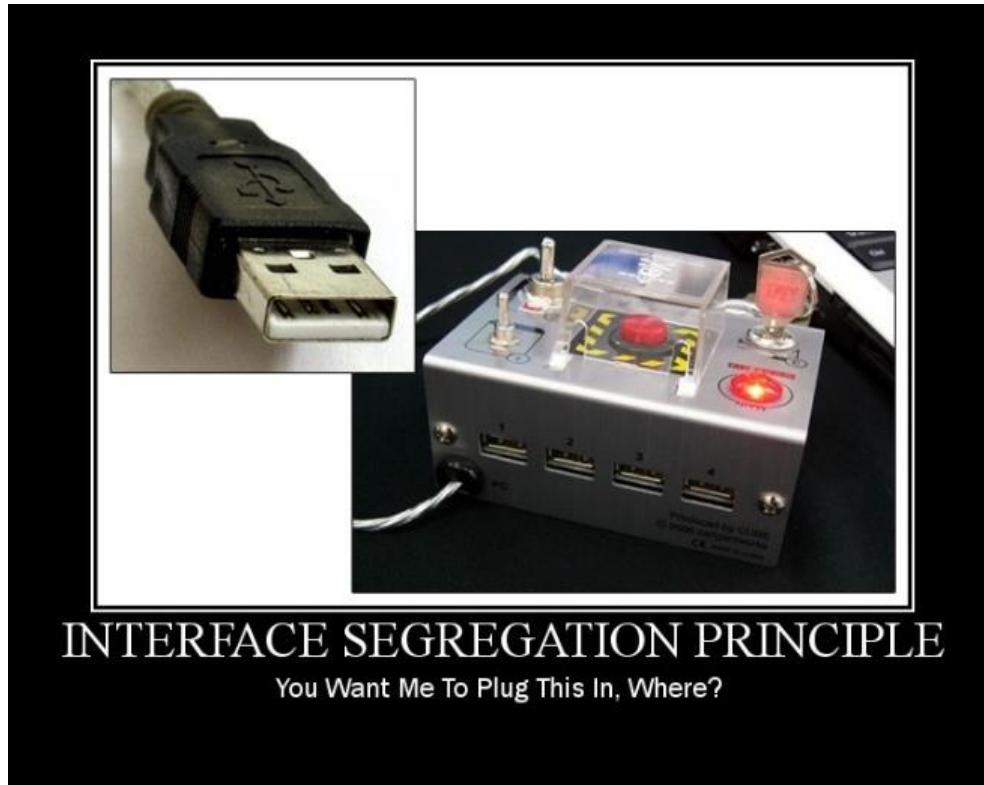


Interface Segregation Principle (ISP)

Interface Segregation Principle (ISP)

Clients should not be forced to depend on methods they do not use

Interface Segregation Principle (ISP)



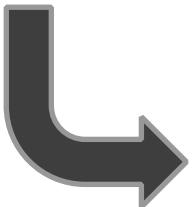
Interface Segregation Principle (ISP)

Why important?

- By following SRP, OCP and LSP, we can build a set of cohesive abstractions enabling reuse in multiple clients
- However, sometimes the abstractions we need are not cohesive (even though they may seem like it at first)
- We need a mechanism for logical separation that still supports combining functions together in a loosely-coupled way
- Otherwise, we'll see "bloating" in our abstractions that can cause unintended/unrelated impact during normal change

Interface Segregation Principle (ISP)

```
7  namespace CoolCompany.Financials
8  {
9      public interface ITaxProcessor
10     {
11         double CalculateSalesTax(double onAmount);
12         double CalculatePropertyTax(double onAmount);
13         double CalculateIncomeTax(double onAmount);
14     }
15 }
```



```
6
7  namespace CoolCompany.Financials
8  {
9      public interface ITaxProcessor
10     {
11         double Calculate(double onAmount);
12     }
13
14     public interface ISalesTaxProcessor : ITaxProcessor
15     {
16         double LookupStateTaxRate(string state);
17     }
18
19     public interface IPropertyTaxProcessor : ITaxProcessor
20     {
21         double LookupTownshipTaxRate(string township);
22     }
23
24     public interface IIIncomeTaxProcessor : ITaxProcessor
25     {
26         double CalculateBackTaxes(string taxpayerId);
27     }
28 }
```

Interface Segregation Principle (ISP)

How to practice?

- In your abstractions, don't force functions together that don't belong together (or that you might want to use separately)
- Leverage delegation in the implementation of those abstractions to support variance
- Use OCP to bring together additional sets of features in a cohesive way (that still adheres to SOLID)
- Use existing tests (or build new ones) to verify aggregate features

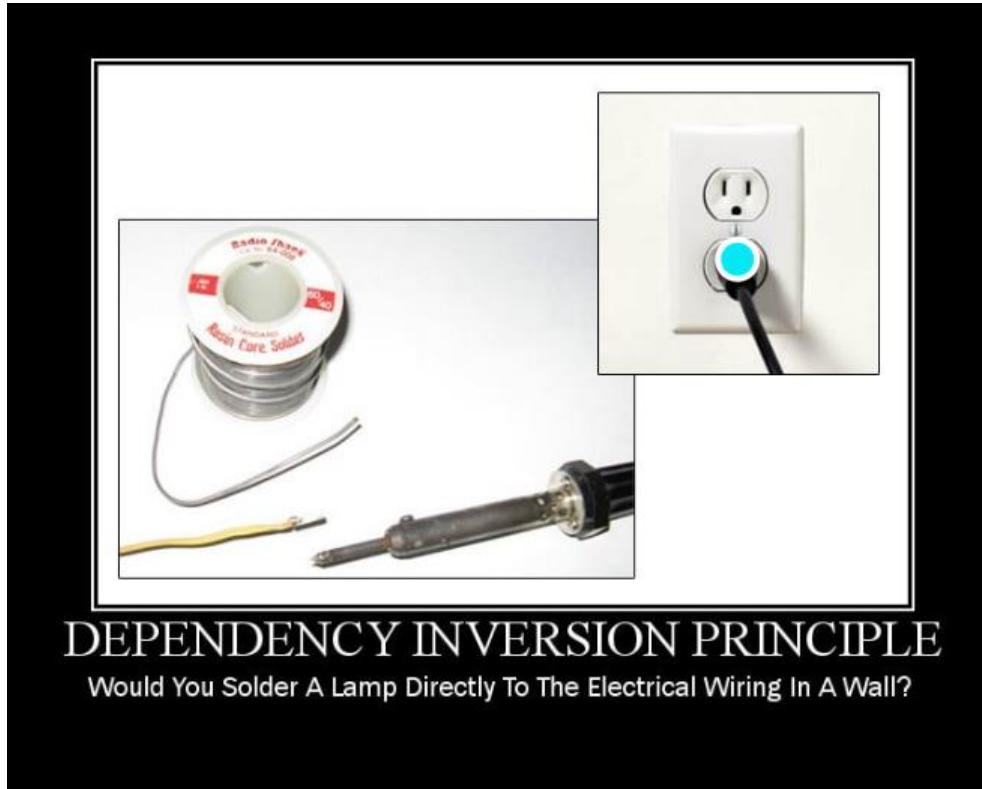
Dependency Inversion Principle (DIP)

Dependency Inversion Principle (DIP)

High-level modules should not depend on low-level modules – both should depend on abstractions

Abstractions should not depend upon details – details should depend upon abstractions

Dependency Inversion Principle (DIP)



Dependency Inversion Principle (DIP)

Why important?

- As with all the SOLID principles, we want to build reusable but loosely-coupled code modules
- We don't want to limit reuse to lower-level utility classes only
- If higher-level modules are tightly-coupled to and dependent on low-level modules, change impact can cascade up
- The change impact can be transitive (flowing through multiple layers in-between)
- There are patterns and utility libraries built to enable

Dependency Inversion Principle (DIP)

How to practice?

- As with the other principles, use (and depend on) abstractions
- Use mechanisms like dependency injection and IoC (Inversion of Control) to build looser coupling between logic providers and consumers
- Build layering into your architectures and limit references to the same or immediately adjacent layer only
- Keep ownership of abstractions with the clients that use them or, even better, in a separate namespace/library
- Use existing tests (or build new ones) to verify functionality in each layer and use mocking techniques to isolate testing



The Twelve-Factor App

The Twelve-Factor App

- Methodology for building applications for the Cloud
- Uses a *declarative* (vs. *imperative*) format for defining setup automation
- Maintains clean contract with the underlying operating system
- Drives toward consistent interface but pluggable implementation based on target OS / platform (see SOLID principles)

The Twelve-Factor App

- Supports modern cloud platforms
- Targets minimal divergence between development and production to enable continuous deployment (agility)
- Targets “elastic scalability” – the ability to dynamically automate scaling of a system to quickly adjust to demand while optimizing cost

<https://12factor.net/>

So, What Does This Have to Do With the Cloud?

- Complex Cloud-enabled workflows are composed of multiple systems
- Those multiple systems are composed of multiple components
- Those multiple components are composed of multiple modules or classes

So, What Does This Have to Do With the Cloud?

- Those multiple modules are composed of multiple routines or blocks of code
- To help ensure quality, testability, stability and maintainability of the complex Cloud-enabled workflows, we must employ good design and build practice in each building block (like a fractal)
- Furthermore, this requires forethought and intentionality – it doesn't happen by accident

Well Architected Framework – Azure

<https://docs.microsoft.com/en-us/azure/architecture/framework/>

Well Architected Framework – AWS

<https://aws.amazon.com/blogs/apn/the-5-pillars-of-the-aws-well-architected-framework/>

Monolith vs. Microservices

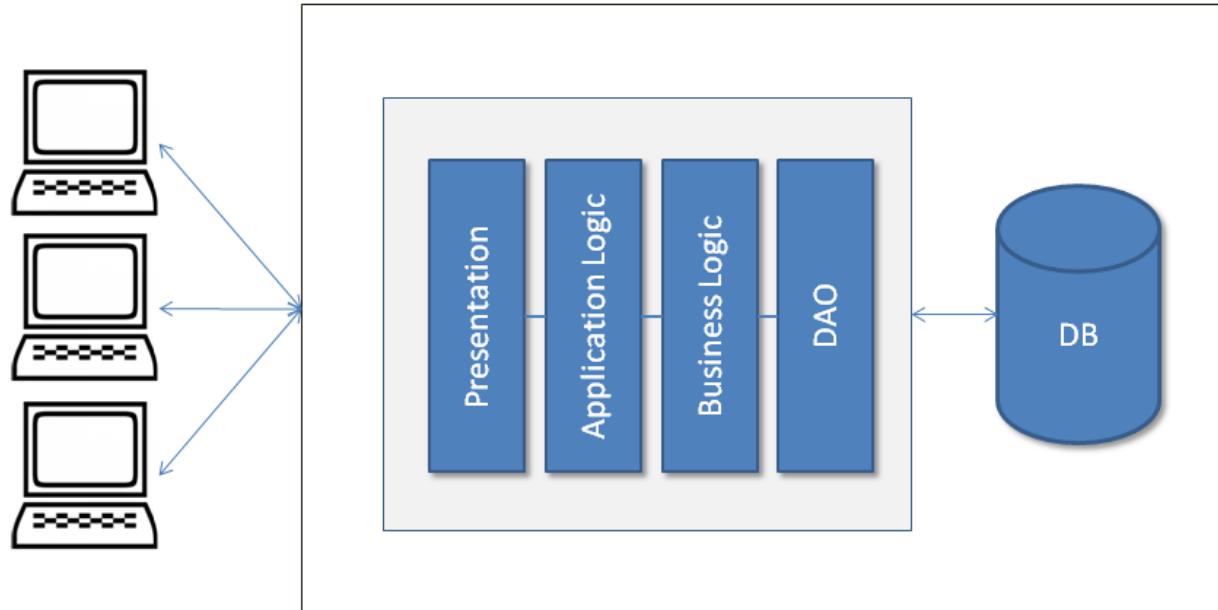
What are Microservices?

- An architectural style in which a distributed application is created as a collection of services that are:
 - Highly maintainable and testable
 - Loosely coupled
 - Independently deployable (by extension, independently-scalable)
 - Domain centric and organized around business capabilities
- The Microservices architecture enables the continuous deployment and delivery of large, complex distributed applications

What is a Monolith?

- Typical enterprise application
- Large codebases
- Set technology stack
- Highly coupled elements
- Whole system affected by failures
- Scaling requires the duplication of the entire app
- Minor changes often require full rebuild

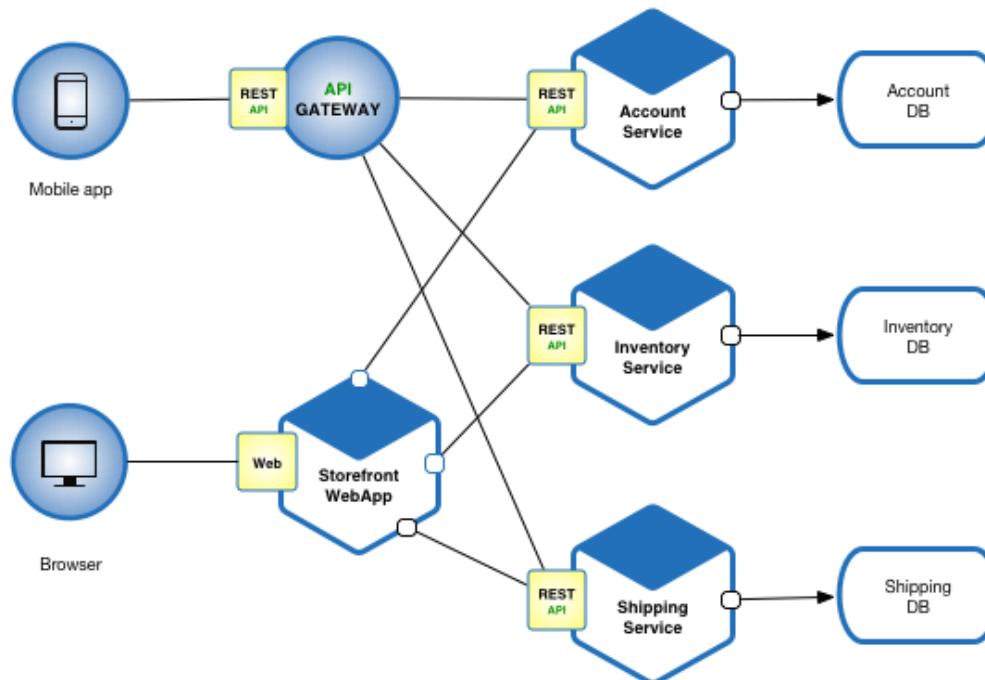
Monolithic Architecture Example



Moving to Microservices

- We have moved towards a refinement of “what is a service?”
- Even in SOA, we can create a service that does more than one thing – using it to solve an overall business problem, rather than one part of the actions needed to solve that problem
 - This was moving us back towards monolithic
- We arrive at a point where we want individual, purposeful services that do one thing and do it well – the microservice

Example Microservices Architecture



Microservices – Benefits vs. Costs

Benefits:

- Enables work in parallel
- Promotes organization according to business domain
- Advantages from isolation
- Flexible in terms of deployment and scale
- Flexible in terms of technology

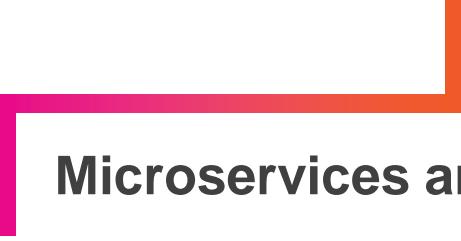
Microservices – Benefits vs. Costs

Costs:

- Requires a different way of thinking
- Complexity moves to the integration layer
- Organization needs to be able to support re-org according to business domain (instead of technology domain)
- With an increased reliance on the network, you may encounter latency and failures at the network layer
- Transactions must be handled differently (across service boundaries)

Microservices & Good Architecture

- With microservices, key concepts:
 - Cohesion (goal to increase)
 - Coupling (goal to decrease)
- SOLID principles & DDD (Domain Driven Design) lay the foundation for:
 - Clean, logical organization of components
 - Maintainability
 - Clear boundaries, encapsulation and separation of concerns in the components used to build out complex systems
 - Techniques that minimize coupling
 - Being “surgical” with our change



Microservices and Containers

What is a Container?

- Loosely isolated environment used to package and deploy an application in a platform-agnostic manner

Evolution of Containers

- Started with client/server architecture – dedicated server for each app
- Moved to Virtual Machines – better, but had problems of OS overhead, etc.
- Containers mean we don't have to worry about specific hardware, OS, language, etc. – we just care about the code

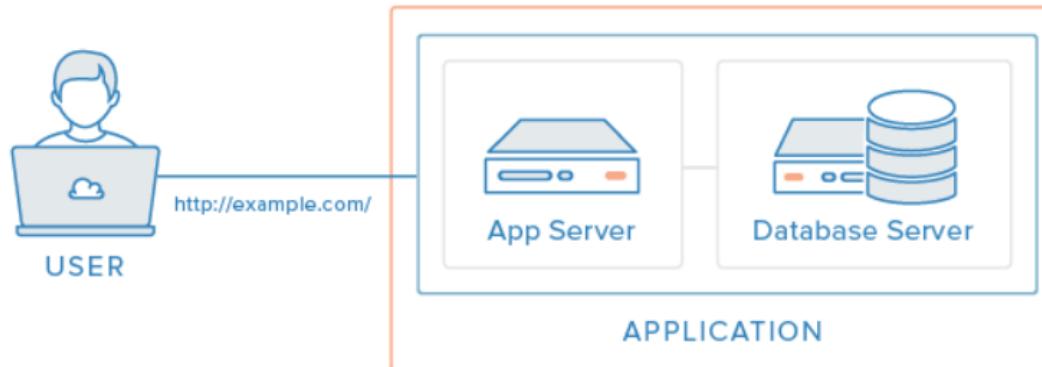
Containers

- Form of virtualization at the app packaging level (like virtual machines at the server level)
- Isolated from one another at the OS process layer (vs VM's which are isolated at the hardware abstraction layer)
- Images represent the packaging up of an application and its dependencies as a complete, deployable unit of execution (code, runtime and configuration)

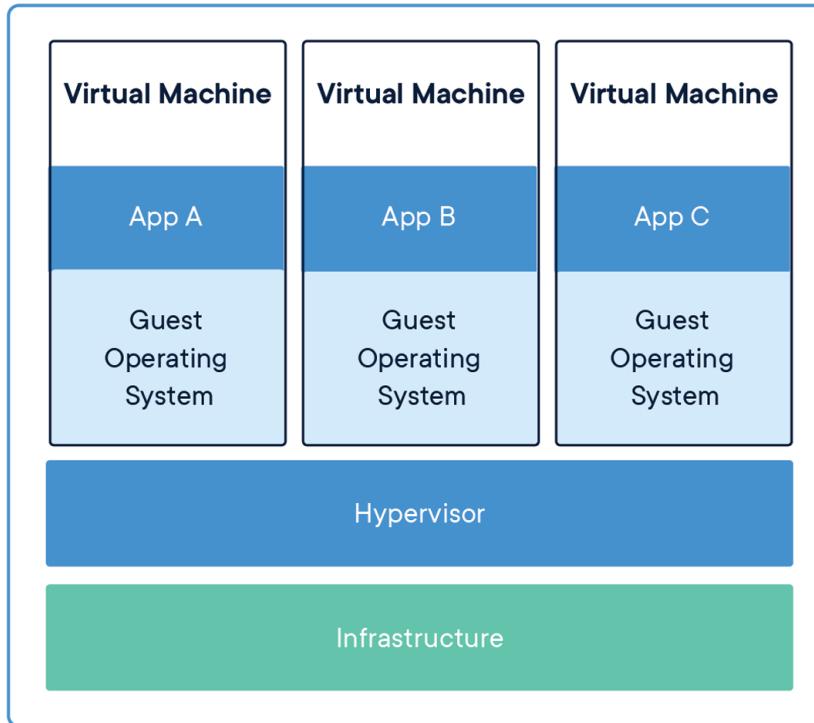
Containers

- A platform (e.g., Docker) running on a system can be used to dynamically create containers (executable instances of the app) from the defined image
- Typically, much, much smaller than a VM which makes them lightweight, quickly deployable and quick to “boot up”
- An orchestration engine (e.g., Kubernetes) might be used to coordinate multiple instances of the same container (or a “pod” of containers) to enable the servicing of more concurrent requests (scalability)

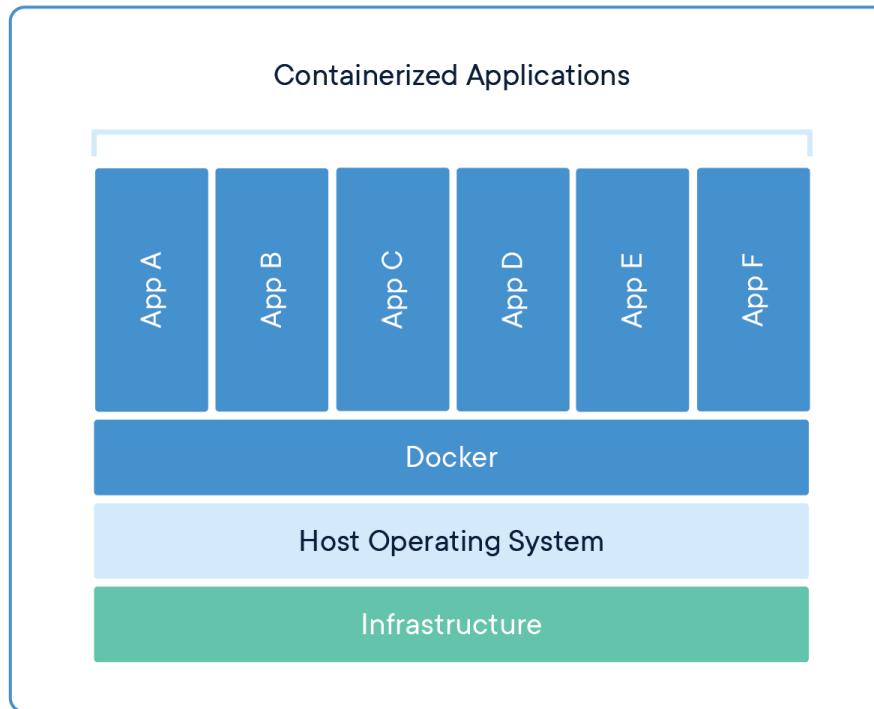
Client/Server



Virtual Machines



Containers



Microservices & Containers

- Microservices – with their smaller size, independently-deployable and independently-scalable profile, and encapsulated business domain boundary – are a great fit for containers
- Using Kubernetes, sophisticated systems of integrated microservices can be built, tested and deployed
- Leveraging the scheduling and scalability benefits of Kubernetes can help an organization target scaling across a complex workflow in very granular ways
- This helps with cost management as you can toggle individual parts of the system for optimized performance

Microservices & the Cloud

- Microservices have broad support across multiple Cloud providers
- One option includes standing up VM's (IaaS) and installing / managing a Kubernetes cluster on those machines or
- Another option includes leveraging a managed service (PaaS) provided by the CSP
- Microservices are a great option for the Cloud because the elastic scalability provided by the Cloud infrastructure can directly support the independent scalability needed with a microservices architecture

Microservices Summary

- Applications that can
 - Efficiently scale
 - Are flexible
 - Are high performance
- These apps are powered by multiple services, working in concert
- Each service is small and has a single focus
- We use a lightweight communication mechanism to coordinate the overall system
- This ends up giving us a technology agnostic API

Demo: Microservices Application

Reference Application: <https://docs.microsoft.com/en-us/dotnet/architecture/microservices/multi-container-microservice-net-applications/microservice-application-design>

Repo: <https://github.com/dotnet-architecture/eShopOnContainers>



Lab

Migration vs. Modernization

Application Portfolio Assessment

- Helps with planning for the migration of an enterprise's system and software estate to the Cloud
- During this process, applications created or utilized by the enterprise will be reviewed for migration disposition
- Assessment may be completed for a small subset of the apps (as a starting point or proof of concept) or may include the entire estate

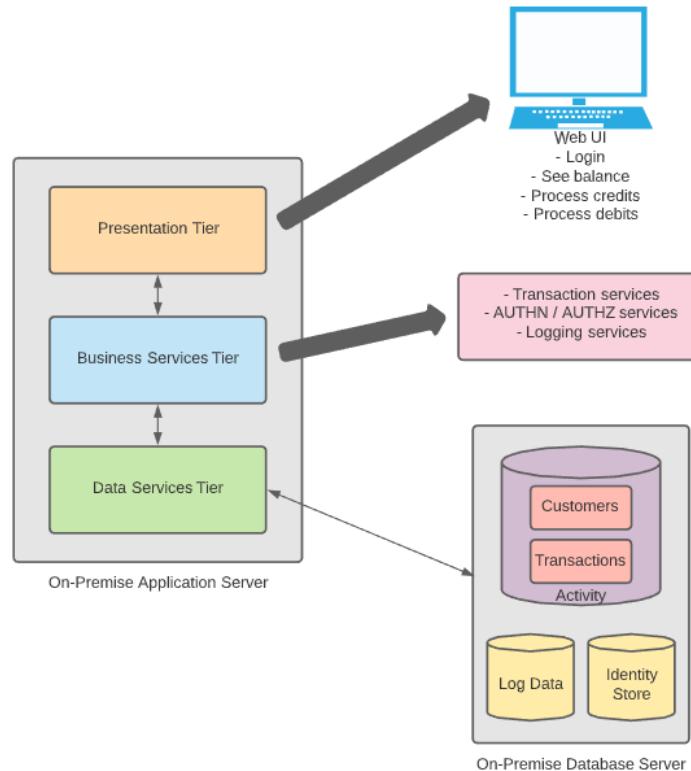
Application Portfolio Assessment

- Through this analysis, a disposition for Cloud readiness will be assigned to each app (based on several factors which we'll review shortly)
- There may also be a determination of which apps are best suited to move first, on the way to defining a plan for migration of all apps in waves
- The assessment will likely include an interview with the application owner (to gain SME knowledge about the application)

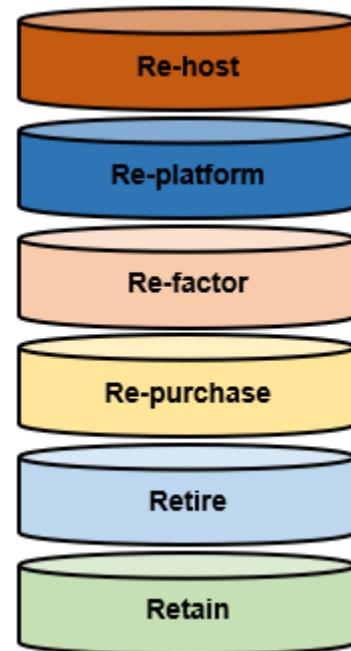
Application Portfolio Assessment

- Will likely include
 - Review of the current state architecture
 - Discussion with stakeholders about roadmap and plans for target state
 - Discussion on expected timing
- The goal is to ensure that migration efforts are focused on the activities that will bring greatest business benefit against optimized cost

Banking App – Current State



The 6 R's – Application Migration Strategies

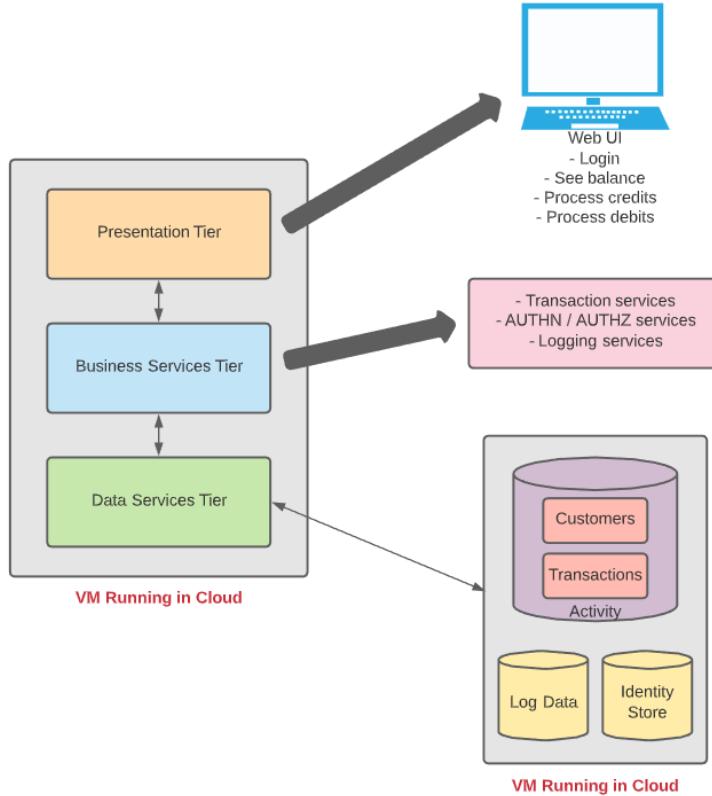




- AKA “lift & shift”
- For all intents and purposes, involves recreating the on-premise infrastructure in the Cloud
- Sometimes used to expedite retirement of a data center



- Can be a mechanism to quickly migrate workloads and see immediate cost savings, even without Cloud optimizations
- There are third-party tools available to help automate the migration
- Once the application is in the Cloud, it can be easier to apply Cloud optimizations vs. trying to migrate and optimize at the same time

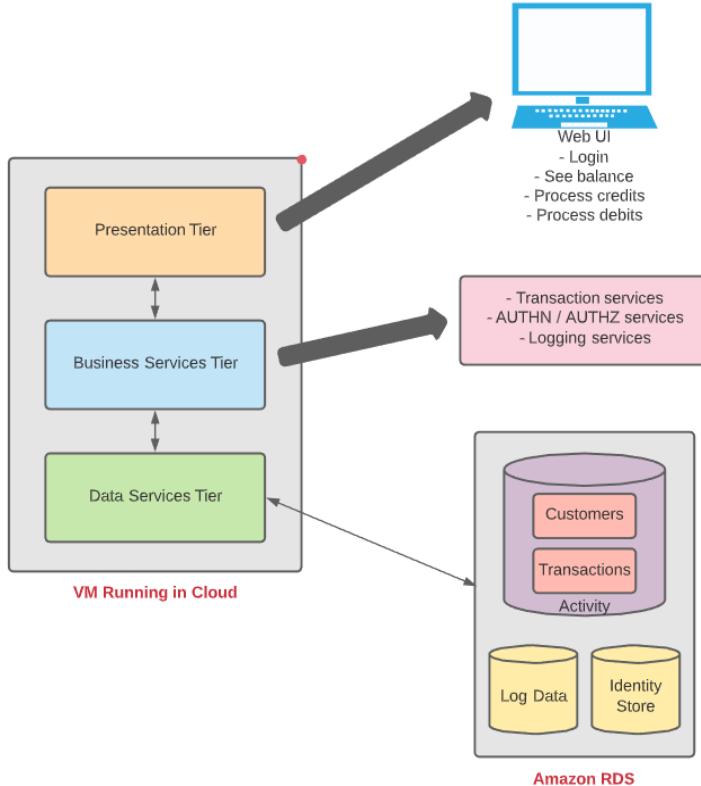


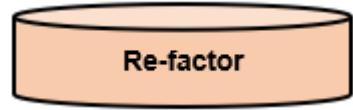


- AKA “lift, tinker & shift” or “lift & fiddle”
- Core architecture of the application will not change
- Involves recreating most of the on-premise infrastructure in the Cloud with a few Cloud optimizations



- Those optimizations will involve a move to one or more Cloud native services for a specific, tangible business benefit
- A common example is moving to a managed database (e.g. Relational Database Service, or RDS, in AWS)
- Enables migration speed while providing cost savings or benefit in a targeted portion of the application's architecture

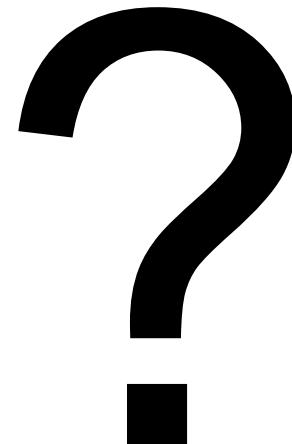




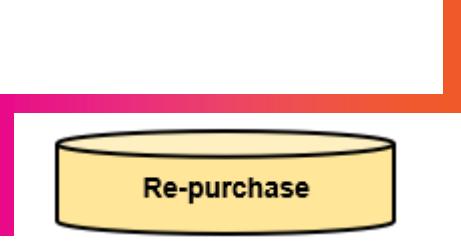
- Involves rearchitecting the application to maximize utilization of Cloud native optimizations
- Likely the most expensive and most complex of the available options
- The application profile needs to fit, and business value must be identified commensurate with the cost required to execute



- Good option if the application can benefit from features, scalability or performance offered by the Cloud
- Examples include rearchitecting a monolith to microservices running in the Cloud or moving an application to serverless technologies for scale



To Be Determined



Re-purchase

- Good fit for applications that are candidates for moving from on-premise licensing (for installed products) to a SaaS model
- Often applications that have been installed to manage a specific type of business capability
- Examples can include Customer Relationship Management (CRM), Enterprise Resource Planning (ERP), HR or e-mail (among others)



- In some cases, an enterprise may have a “healthy” percentage of legacy applications that are no longer being used or maintained (especially for larger organizations)
- When completing the Application Portfolio Assessment and associated interviews with application owners, look for opportunities to recommend retire of the unused legacy apps
- This type of application can represent a “quick win”

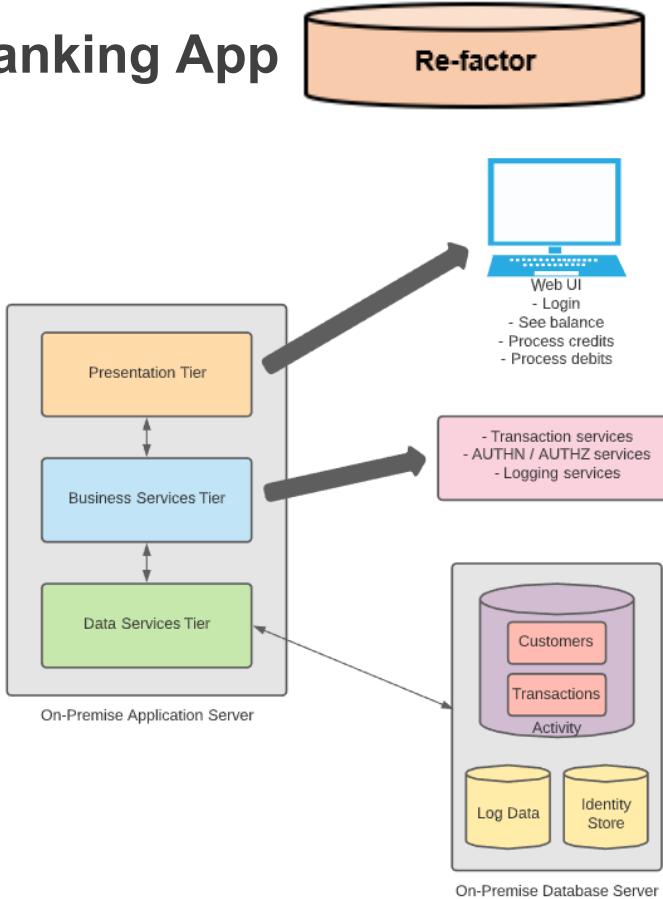


- The associated savings can potentially be advantageously factored into the business case for the Cloud modernization effort
- Finally, retiring unused apps reduces attack surface area from a security perspective

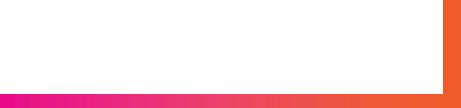


- Applications that must remain in place but that cannot be migrated to the Cloud without major refactor
- This type of application profile may prevent the ability to completely move out of the data center (at least in the interim)
- From a cost perspective, it can be difficult for an enterprise to take on both the operating expense of Cloud while continuing to carry the capital expense of a data center and on-premise infrastructure
- The hybrid model can be a good solution to support where needed

Banking App



If you wanted to sell an investment project to a client for rearchitecting this application for migration to the Cloud, what are some of the potential benefits you might put forth as justification?



Lab

Cloud Services

Cloud Services

- Much of our discussion so far has been on infrastructure – VM's, network, storage, data, etc.
- However, the Cloud offers much more in the way of capability and services
- Potential benefit of these services
 - Readily available on all major Cloud platforms
 - Relatively easy to configure (though not necessarily easy to integrate)
 - Can take advantage of Cloud scale

Cloud Services

- Sophisticated services that bring business value
- Enable expansion of the services ecosystem of your enterprise into new areas of differentiation and segmentation
- Key thing to remember – just because a service is available does not mean that you must use it (or even should use it)
- There should be an architectural vision in place for leverage of the Cloud that is directly aligned with business value

Cloud Services

- Why? Because Cloud services will have a cost associated
- The cost must be accounted for and justified as operating expense against business drivers
- With good alignment and a good plan, sophisticated Cloud services can help accelerate business mission

Cloud Solution Providers (CSPs)

Cloud Solution Providers (CSPs)

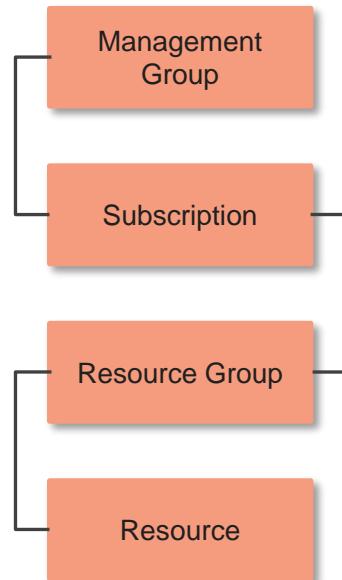
- Other players in the space:
 - Microsoft Azure
 - AWS (Amazon Web Services)
 - GCP (Google Cloud Platform)
- We're going to focus on comparing Azure & AWS

Microsoft Azure

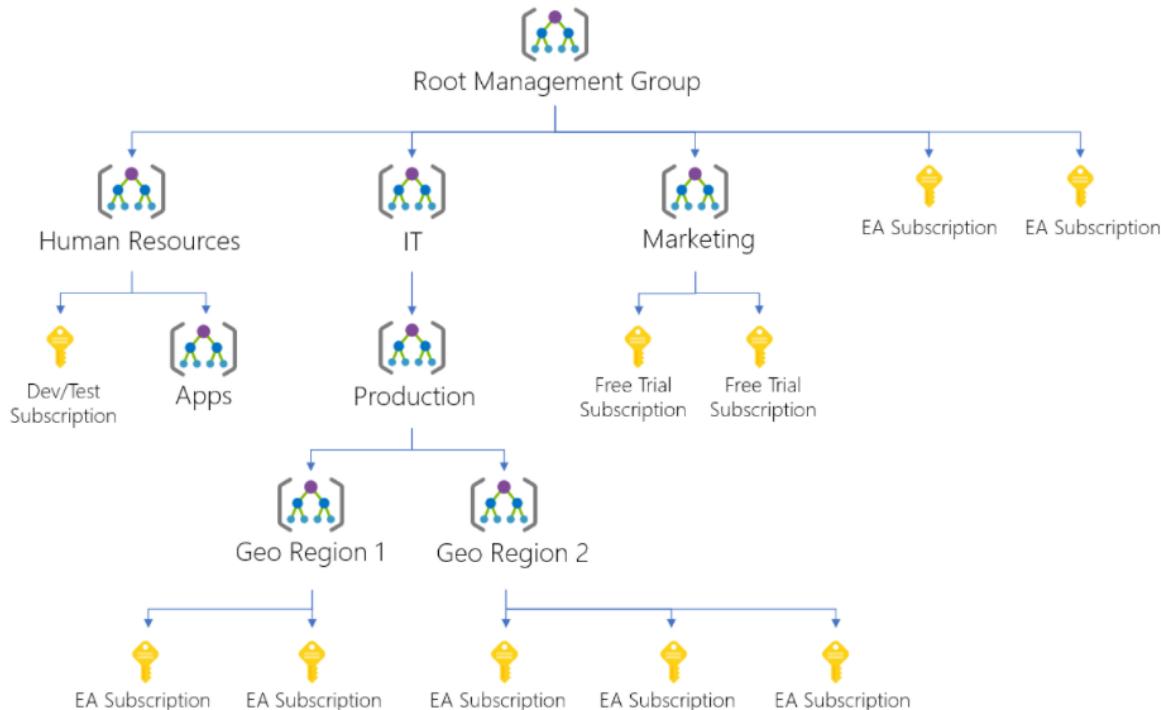
- Portal accessible at <https://portal.azure.com>
- Identity & Access Management handled via Azure Active Directory
- Users, groups, roles, and policies
- Unit of access & activity is the “subscription”

Microsoft Azure

Additional layer of management group sits above subscription



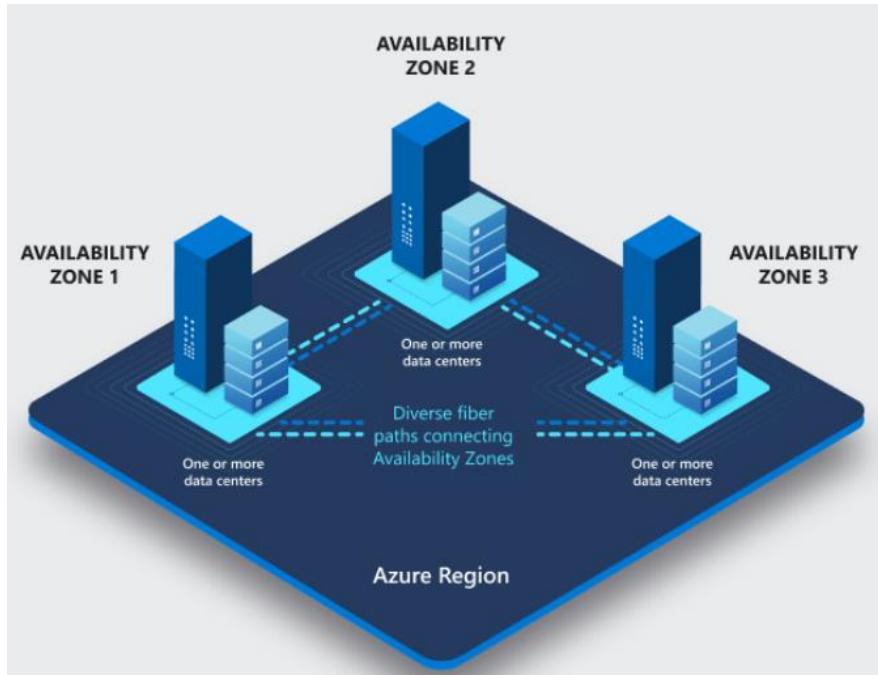
Microsoft Azure



Microsoft Azure

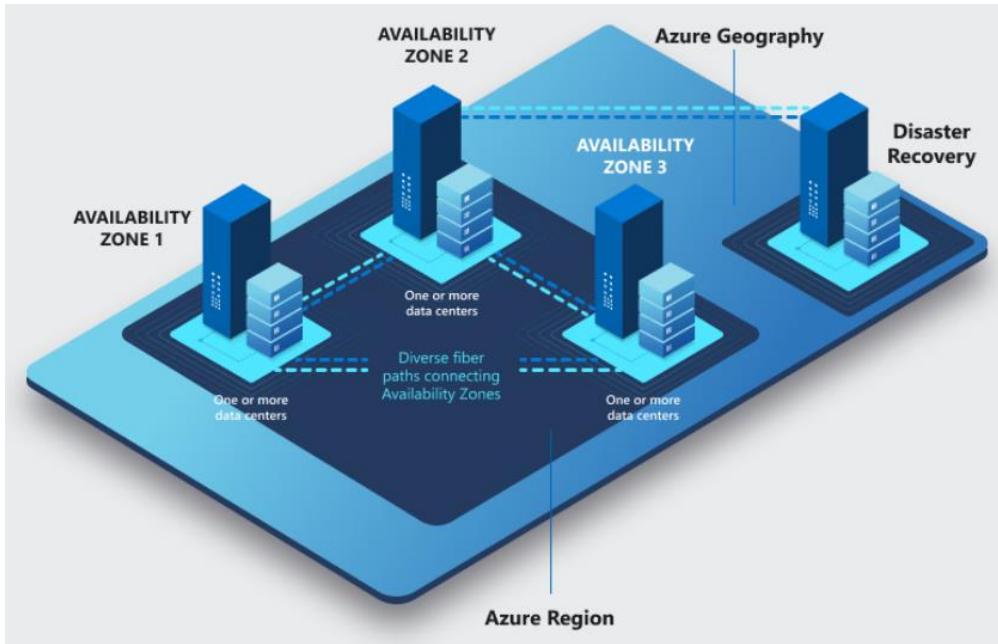
- Uses concept of “region” for geographical location of resources
- By selecting region closest to app usage, can minimize latency
- Each region includes multiple availability zones (clustering of data centers in that region)
- And region pairs provide support for BC/DR

Microsoft Azure



- Intended to help protect against data center failure
- Utilizes private fiber for optimized network connectivity

Microsoft Azure

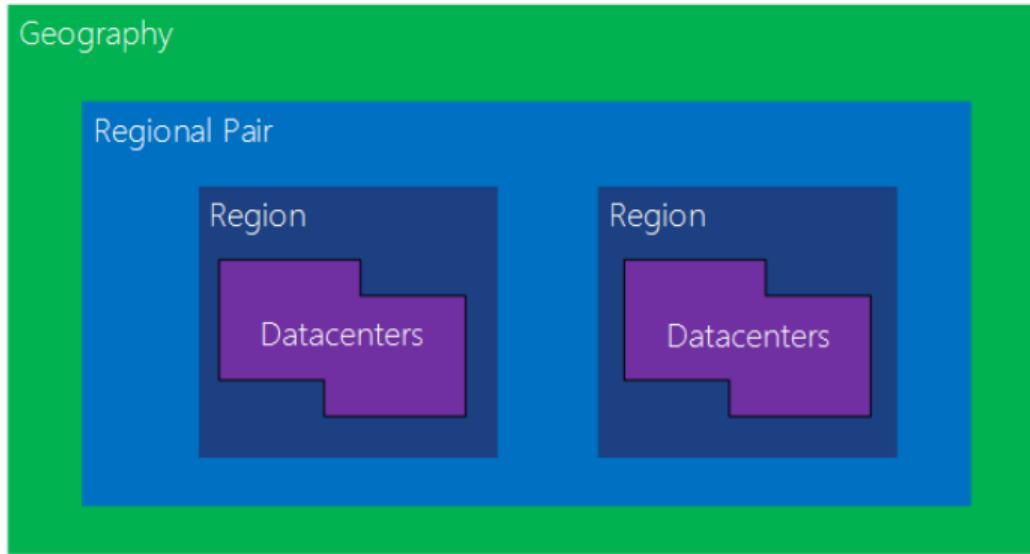


- Regions are paired
- Maintenance on pairs is coordinated to help minimize downtime

Microsoft Azure

Americas	Europe	Africa	Asia Pacific
Brazil South	France Central	South Africa North	Australia East
Canada Central	Germany West Central		Central India*
Central US	North Europe		Japan East
East US	Norway East		Korea Central
East US 2	UK South		Southeast Asia
South Central US	West Europe		East Asia*
US Gov Virginia	Sweden*		
West US 2			
West US 3			

Microsoft Azure



Microsoft Azure

Examples of region pairs include:

Primary

West US

North Europe

Southeast Asia

Secondary

East US

West Europe

East Asia

Full list available at <https://docs.microsoft.com/en-us/azure/best-practices-availability-paired-regions#what-are-paired-regions>

Microsoft Azure

- When creating Azure resources, organized in resource groups
- Mechanism for organizing access, security, and resource boundaries
- Every Azure resource will live in a resource group

AWS (Amazon Web Services)

- Management console accessible at <https://aws.amazon.com>
- Identity & Access Management handled via the IAM service
- Users, groups, roles, and policies
- Unit of access & activity is the “account”

AWS (Amazon Web Services)

- No real concept of management group
- AWS Control Tower provides similar capabilities
- Assignment of groups & roles to AWS organizations

AWS (Amazon Web Services)

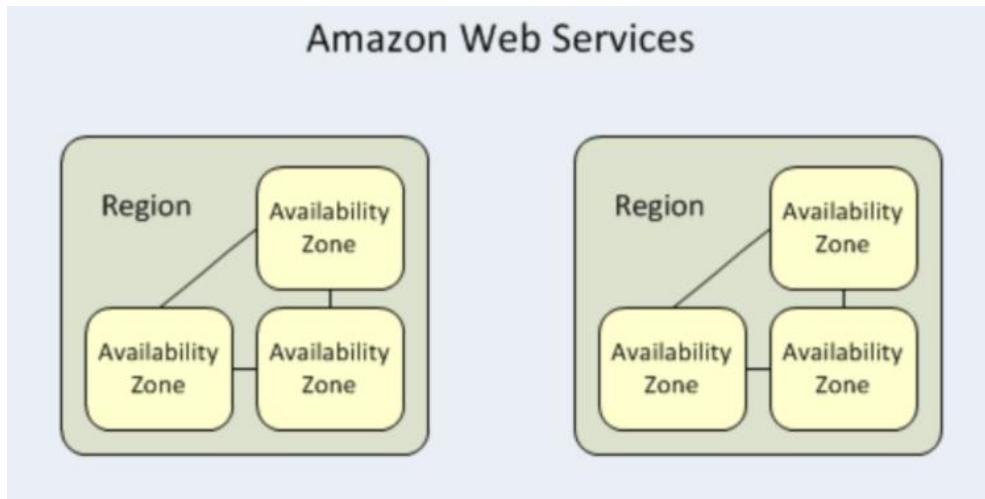


See <https://aws.amazon.com/controlltower> for additional detail (similar to Azure Blueprints)

AWS (Amazon Web Services)

- Also uses concept of “region” for geographical location of resources
- Similar capabilities as with Azure
- Multiple geographies and multiple availability zones within geography
- Does not use concept of pairs but still manages availability (implicit)

AWS (Amazon Web Services)



- Intended to help protect against data center failure
- Utilizes private fiber for optimized network connectivity
- Mirrors Azure

AWS (Amazon Web Services)

- See https://aws.amazon.com/about-aws/global-infrastructure/regions_az/ for additional detail

AWS (Amazon Web Services)

- No real comparable option as with Azure resource groups
- Able to use tags (in Azure also) for organizing/managing resources

Developing for Cloud

Developing for Cloud

- Key areas of focus

- Compute
- Storage
- Networking
- Database

Compute in Azure

- Includes
 - Virtual Machines
 - Azure Logic Apps
 - Azure Functions
 - Azure App Services

Compute in AWS

- Includes
 - EC2
 - Lambda
 - Elastic Beanstalk

Storage in Azure

- Includes
 - Azure Storage

Storage in AWS

- Includes
 - S3 (Simple Storage Service)

Networking in Azure

- Includes
 - Virtual Networks
 - Subnets
 - Azure API Management
 - Azure Traffic Manager

Networking in AWS

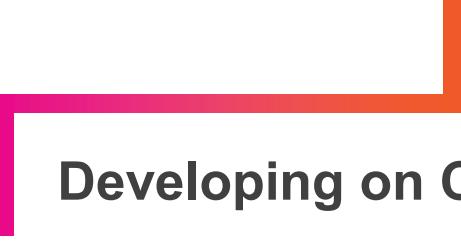
- Includes
 - VPCs
 - Subnets
 - API Gateway
 - Route 53

Database in Azure

- Includes
 - Azure SQL
 - Azure Cosmos DB
 - Azure Cache for Redis

Database in AWS

- Includes
 - RDS
 - Amazon DynamoDB
 - Amazon Aurora



Developing on Cloud

Developing on Cloud - Azure

- Includes
 - Azure Resource Manager (ARM) templates
 - Azure Cloud Shell (includes Cloud-enabled VS Code interface)
 - Azure Blueprints

Developing on Cloud - AWS

- Includes
 - AWS CloudFormation
 - AWS Cloud9
 - AWS Control Tower

Internet of Things (IoT)

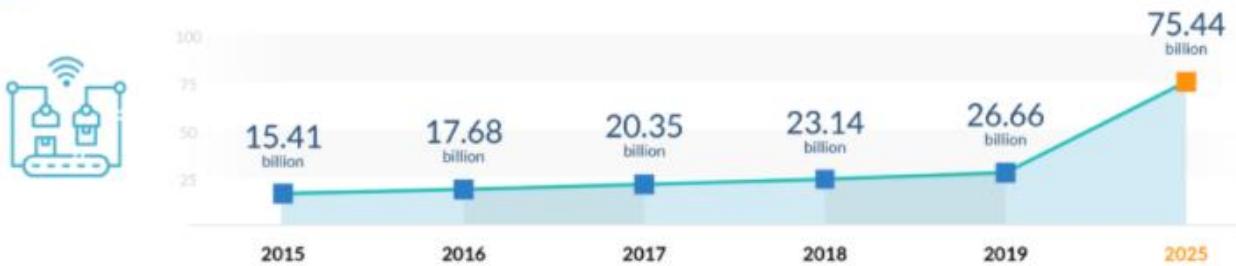
Industry Trends

From <https://financesonline.com/iot-trends>

1

Number of Installed IoT devices around the world

Source: Statista



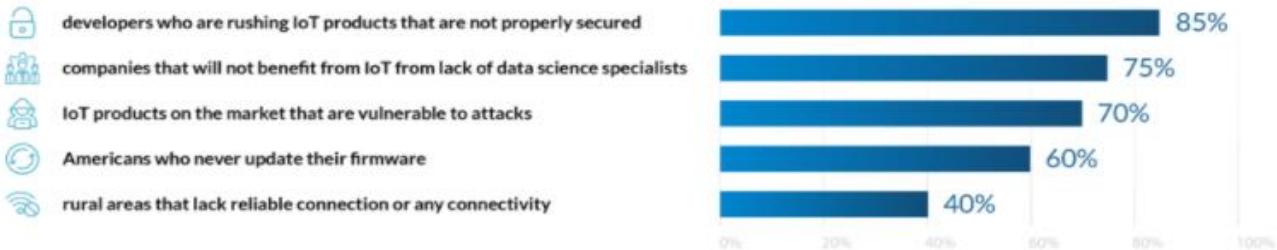
Challenges potentially encountered with that amount of projected growth?

Industry Trends

From <https://financesonline.com/iot-trends>

2 Major challenges IoT technology is facing

Sources: Innovation Enterprise, Gartner, Entrepreneur Media, Bitdefender, Brookings Institution



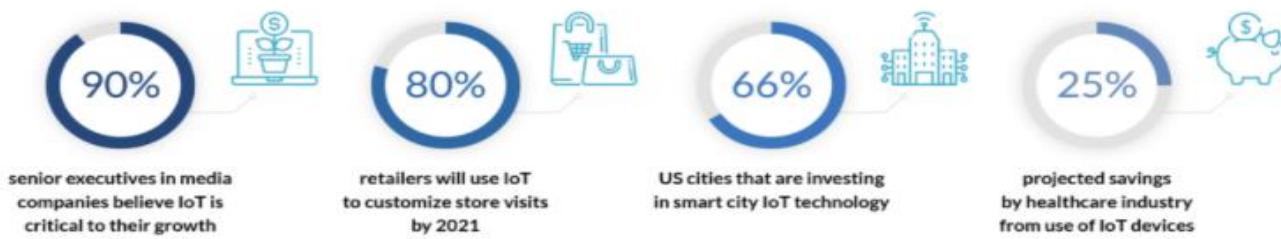
Industry Trends

From <https://financesonline.com/iot-trends>

3

Perceived, expected, and real benefits of IoT

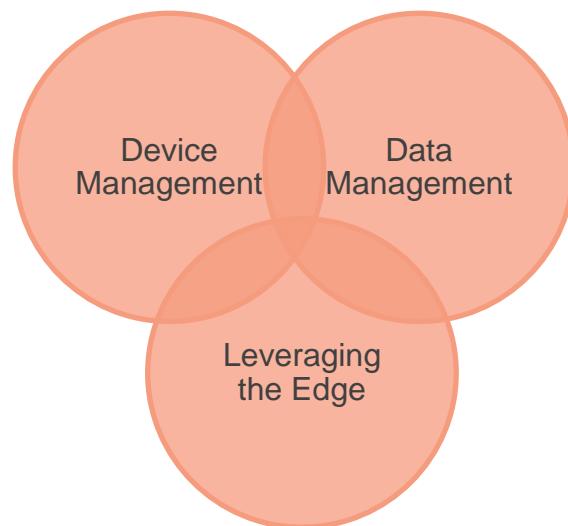
Sources: Statista, SAS, Data Smart City Solutions, Tech Republic, Health IT Analytics



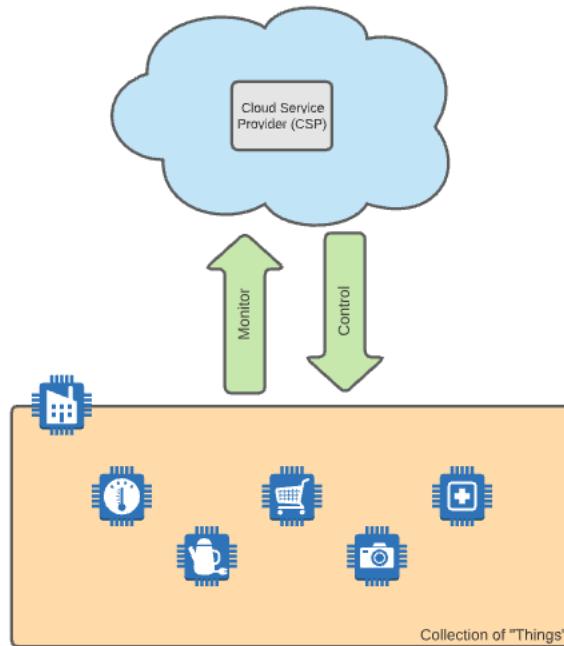
Industry Trends

Areas of focus:

- Accommodating orders of magnitude of growth
- Preventing known (or unknown) challenges from impeding success
- As a business, effectively leveraging opportunities to meet customer needs



Cloud-Centric IoT (CIoT)



Cloud-Centric IoT (CloT) – Potential Issues

Bandwidth

- Data size
- Data frequency

Latency

- Impact of processing delays
- Inability to react in “real-time”

Need for “Always On”

- Long transmission distances
- Intermittent network connectivity

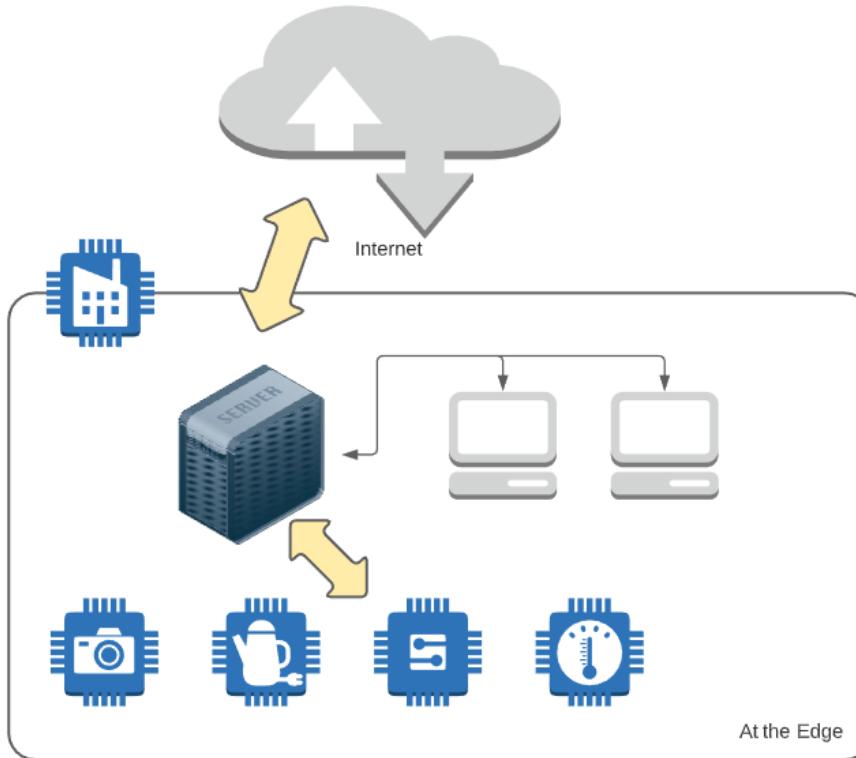
Resource Constraints

- Not enough on-device power for processing complexity
- End-to-end transmission consumes precious energy

Security

- Sometimes remote and outside
- Insufficient on-device capability to prevent service interruption or data “spoofing”

At the Edge



Fog & Edge Computing

At its simplest, involves the injection of intermediate components:

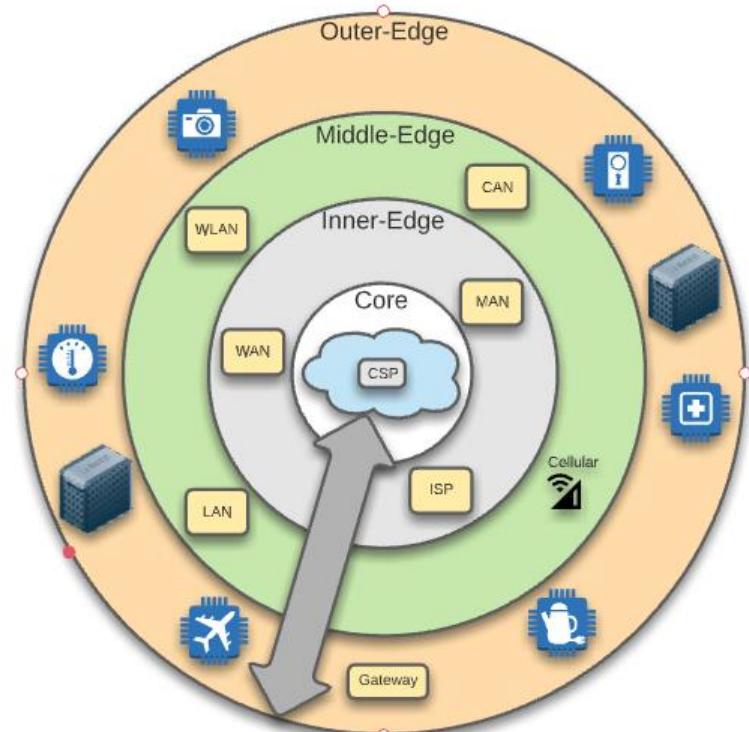
- Bring more power closer to the “things”
- Offload processing
- Optimize what gets transmitted
- Caching to remediate intermittent connectivity issues
- Supplement with additional layers of security

Fog & Edge Computing

Can include three distinct layers:

- Inner or near edge
- Middle edge
- Outer or far edge

At the Edge – Layers



CSP – Cloud Service Provider
WAN – Wide Area Network
ISP – Internet Service Provider
MAN – Metropolitan Area Network
LAN – Local Area Network
WLAN – Wireless Local Area Network
CAN – Campus Area Network

Internet of Things (IoT)

- Supports the creation of a system of interconnected devices that can automatically transfer data of note over a network
- Examples include things like:
 - Temperature & humidity sensors in a data center to monitor climate
 - Sensors in remote oil fields used to monitor status of equipment without the costly requirement to have someone travel to the site to gather status
 - Sensors embedded into hand sanitizing stations that can notify a central management facility when close to empty (for efficient refill)
 - Security cameras that read data (e.g., facial recognition or GPS location) and pass to a central hub for tracking or triangulation
- The devices can be very sophisticated or very simple – depending on use case

Internet of Things (IoT)

- Data pulled by the device can be event driven or polled on a schedule
- Can generate massive amounts of data – the Cloud services previously covered are beneficial to gathering and harnessing intelligence in the data
- By leveraging Cloud scale, IoT systems can
 - Process data from all over the world
 - Quickly ingest it
 - Leverage Machine Learning to provide predictive analytics or predictive diagnostics

Internet of Things (IoT)

- Depending on the firmware (low-level coding at the hardware layer), devices may support both monitor and control
- Given the often-remote nature of the devices (and, therefore, less control over physical security), IoT systems should be built with security at top of mind

Fog & Edge Computing – Advantages

Security

Cognition

Agility

Latency

Efficiency

Fog & Edge Computing – S.C.A.L.E.

Security:

- Additional layers of protection
- Can support security patches to devices in remote areas
- Improves speed with which patches can be applied

Fog & Edge Computing – S.C.A.L.E.

Cognition:

- More processing power closer to the device
- Means more sophisticated decision-making (“smart devices”)
- Can include components that enable self-healing
- Provides options for even simplest of “things”

Fog & Edge Computing – S.C.A.L.E.

Agility:

- Provides additional “levers” for configuration and operation
- Enables commoditization of intermediate layer capabilities
- Supports reuse of open software interfaces and SDK’s

Fog & Edge Computing – S.C.A.L.E.

Latency:

- Improves speed of response to and from the “things”
- Optimizes communications by localizing aggregation and processing
- Data travels faster over shorter routes

Fog & Edge Computing – S.C.A.L.E.

Efficiency:

- Caching and batching can improve overall performance
- Data analysis can be completed closer to the device
- Can generate results faster by execution closer to the data source
- Cloud communication still available but data transmitted can be filtered

How Are Advantages Realized?

Storage

- Data caching & temporary storage
- Data stability & consistency even with network “blips”

Compute

- Hardware & software for localized execution of logic
- Multiple hosting options (including containerization)

Acceleration

- Networking acceleration – network virtualization and software-defined network (SDN)
- Computing acceleration – vertical and horizontal scaling

Networking

- Vertical networking – efficient communication across the edge layers
- Horizontal networking – efficient communication within the edge layers

Control

- Control over deployment
- Control over device behavior & configuration
- Control over management of disparate protocols
- Control of security

Case Study

Case Study is focused on modernization and digitization of an in-home nursing assistance service.
Current state:

- Service maintains in-home care with in-person, routine visits every 4 days
 - ~400 patients within a 125-mile radius of a major metropolitan area
 - Routine visits include checking vitals, delivery of medication, and, in some cases, special care
 - Average age of patients is 71+ years old
 - Patients will not be able to administer any IT infrastructure installed in the home
 - Network connectivity can be inconsistent at times with intermittent, temporary drops
 - Historical data tracking is important to assess quality of patient care and identify potential improvements
 - HIPAA regulations (health privacy) dictate a need for responsible stewardship of the data
-
- ✓ What questions would you ask to gain additional requirements information?
 - ✓ If you were going to implement an Edge/IoT solution for this case study, what major software/hardware components might you include?
 - ✓ What kind of challenges do you anticipate encountering with implementation? What are some key things to “keep in mind”?

Device Management

Device Management

- To be secure, IoT systems must manage all points of data ingress and egress
- Unknown devices should remain restricted
- Need a repeatable process for onboarding/offboarding devices

Device Management

- In addition to onboarding, mechanisms for device maintenance required
- Firmware updates, security patches, reboots/resets
- Supported by a structured lifecycle

Device Management

- In many cases, must accommodate hundreds or thousands of devices
- May include devices located miles away or in remote areas
- Means manual configuration not always feasible

Device Management

Principles



Scale & Automation

- Keep human-to-device ratio cost effective (bulk processing)
- Automate as much as possible
- Enable secure remote management techniques over distance

Openness & Compatibility

- Device ecosystem includes multiple variants
- Can include different profiles, platforms, and protocols
- Requires strategies that:
 - Can accommodate the variants
 - Unify where possible from reporting and support perspectives

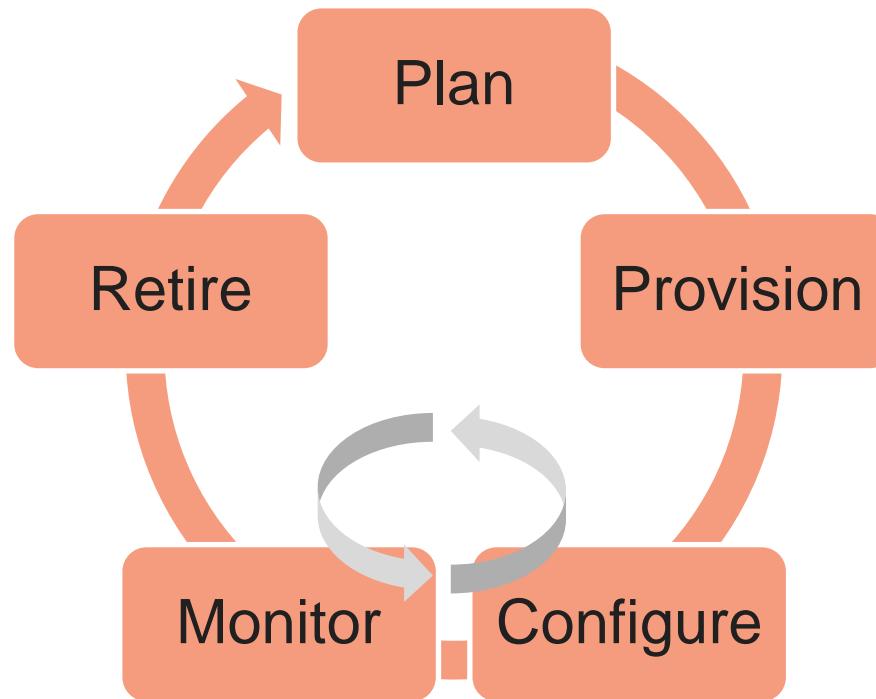
Context Awareness

- IoT environments need to maintain awareness of critical factors like:
 - SLAs for maintenance windows
 - Network and power states
 - Geo-location information (in some applications)
- In some systems, lack of awareness can be damaging or dangerous

Service Many Roles

- Operational support requirements must be understood
- Reporting & dashboarding are key
- Need ways to tailor detail to different types of stakeholders

Device Lifecycle

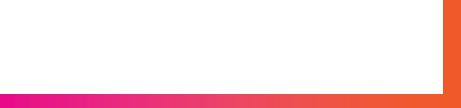


Case Study

Your team has been hired to design and implement an autonomous vehicle powered by Edge & IoT. High-level requirements are as follows:

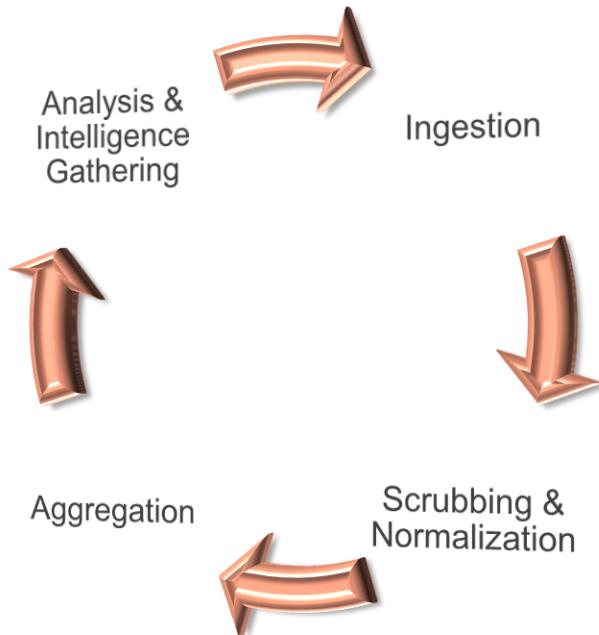
- Standard routes must be supported – point A to point B
 - Turns, lane changes, regular traffic stops must occur safely
 - If weather forecast for an area the vehicle is about to enter indicates treacherous driving conditions, the vehicle should automatically adjust in speed, stopping distance, etc.
 - If the vehicle is approaching an area where there is significant construction, the vehicle should find an alternate, more efficient route but must confirm with the user before switching to it
 - The user should be allowed to take control of the vehicle at any time
 - Information about miles, speeds, maneuvers, weather conditions, and road conditions should be aggregated to a central location for analysis and Machine Learning to foster continuous improvement in safety and utility
-
- ✓ What major software/hardware components would you consider including?
 - ✓ What would be the “devices” in this scenario?
 - ✓ What would device registration, configuration, monitoring, and retirement look like in this scenario from an Edge/IoT perspective?

Demo



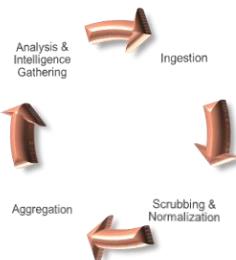
Data Management

Data Management – Stages



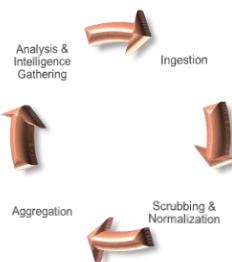
Data Ingestion

- Could be via message exchange or streaming
- Depending on size/scope, may translate to **LARGE** amounts of incoming data



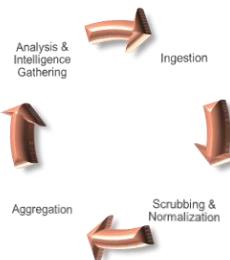
Data Ingestion

- Because of potential scale, bandwidth may be a concern
- Depending on application, latency may also be a concern
- Data may require translation (e.g., from low-level bytes to object or JSON)



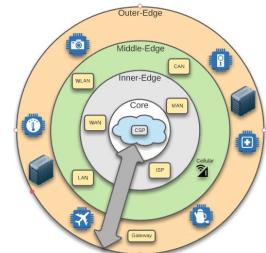
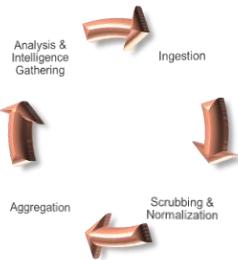
Data Ingestion

- Event hubs or streaming analytics platforms support ingestion at scale
- Provide time and context-aware processing for correct sequencing
- Data may flow through intermediate storage on way to final processing
- Depending on sensitivity of data, could require robust security at each stop



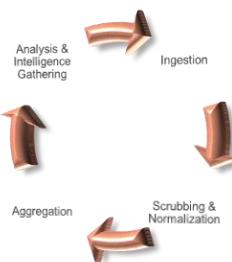
Data Ingestion – What About the Edge?

- Edge components (e.g., gateways) can help optimize
- Preliminary processing at the edge can be used to filter what really matters
- Potential for bundling or compressing data for transmit to cloud
- Can help with bandwidth or latency issues



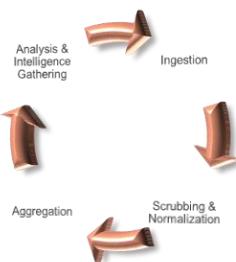
Data Scrubbing & Normalization

- Depending on payload, some portions of the data may not be needed
- Or some portions might contain sensitive detail
- Those parts not needed or sensitive can be “scrubbed” to exclude



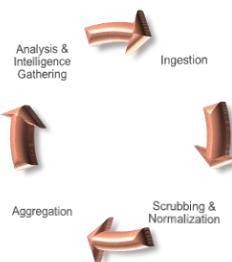
Data Scrubbing & Normalization

- Represents another potential optimization that can preserve storage
- In other cases, similar data may be coming in multiple, disparate formats



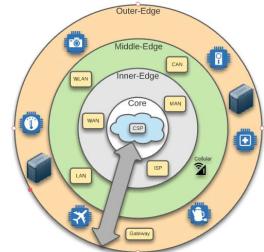
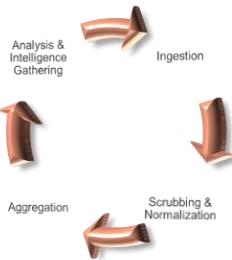
Data Scrubbing & Normalization

- Normalization can bring consistency to the disparate content
- By normalizing, becomes a single dataset for comprehensive analysis
- Normalization may happen as part of ingestion or as part of a separate step



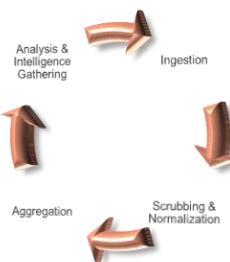
Data Scrubbing & Normalization – What About the Edge?

- Depending on complexity, may execute faster closer to the data
- Might involve proprietary algorithms best kept within full control
- Allows addressing of sensitive data before routed to Cloud
- Can also provide additional optimization (relative to bandwidth)



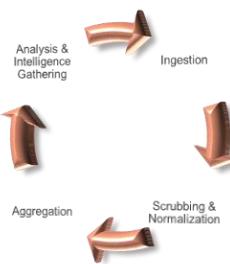
Data Aggregation

- Helps provide full picture of data from multiple streams
- May also be used to enrich with info from other data sources
- Data will be stored in persistent storage for downstream analysis & reporting



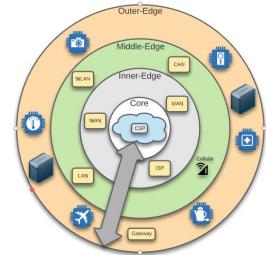
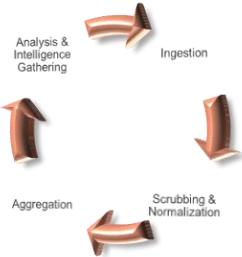
Data Aggregation

- In statistical analysis, the larger the sample size, the more accurate the inference
- To manage costs, large sets of data may leverage different types of storage:
 - Hot storage – most recent data and most relevant for current analysis
 - Cool storage – data not actively used but potentially relevant (short-term trends)
 - Cold or archive storage – data kept for historical purposes and long-term trending
- Security of the stored data and encryption at rest become critical



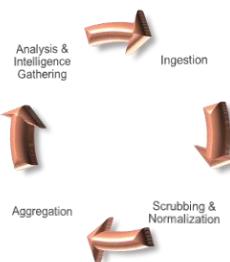
Data Aggregation – What About the Edge?

- Provides an additional layer of storage
- Data not transmitted to Cloud (due to optimizations) may still be valuable to keep
- Enables storage of sensitive data in “raw” format in controlled environment
- Can help balance costs against short to mid-term retention requirements



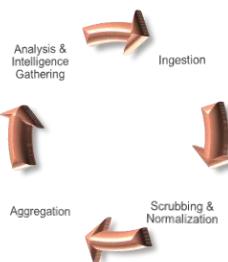
Data Analysis & Intelligence Gathering

- In the digital age, data is the competitive edge
- Companies that manage their data as a critical asset succeed
- Keys:
 - Aggregating efficiently
 - Analyzing effectively



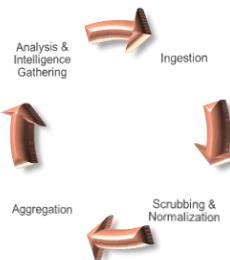
Data Analysis & Intelligence Gathering

- Goal is to identify and leverage the most important data points
- Importance is measured by business value-driven decision-making
- What can I learn about today's customers, scenarios, or business cases?
- What can I effectively predict about tomorrow?



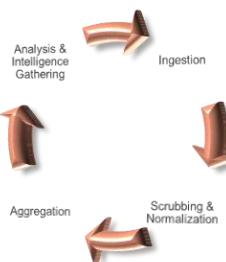
Data Analysis & Intelligence Gathering

- Requires balancing of competing concerns:
 - To increase quality of intelligence, more data is required (sometimes MUCH more)
 - But massive datasets can be complex to manage and process



Data Analysis & Intelligence Gathering

- Enter ML / AI:
 - Algorithms are used to build mathematical models from existing data
 - Results in a mathematical “trajectory” (and confidence level)
 - Algorithms can be configured to learn and improve over time
- Hyperscale available in the Cloud brings near-limitless power to bear



Machine Learning / Artificial Intelligence

- Technology and computer systems are phenomenal at “crunching” large amounts of data
- When Cloud-enabled with access to the scale of the Internet, the amount of data that can be processed and the complexity of the “crunching” can increase severalfold
- Machine Learning is considered a subset of Artificial Intelligence

Machine Learning / Artificial Intelligence

- Involves system algorithms that can improve automatically over time by learning from “experience” (depending on configuration)
- This “experience” comes largely through the aggregation and processing of large amounts of data
- The data provides a view as to what happened in the past
- That information can be used to make “predictions” (or calculated assumptions) about the future

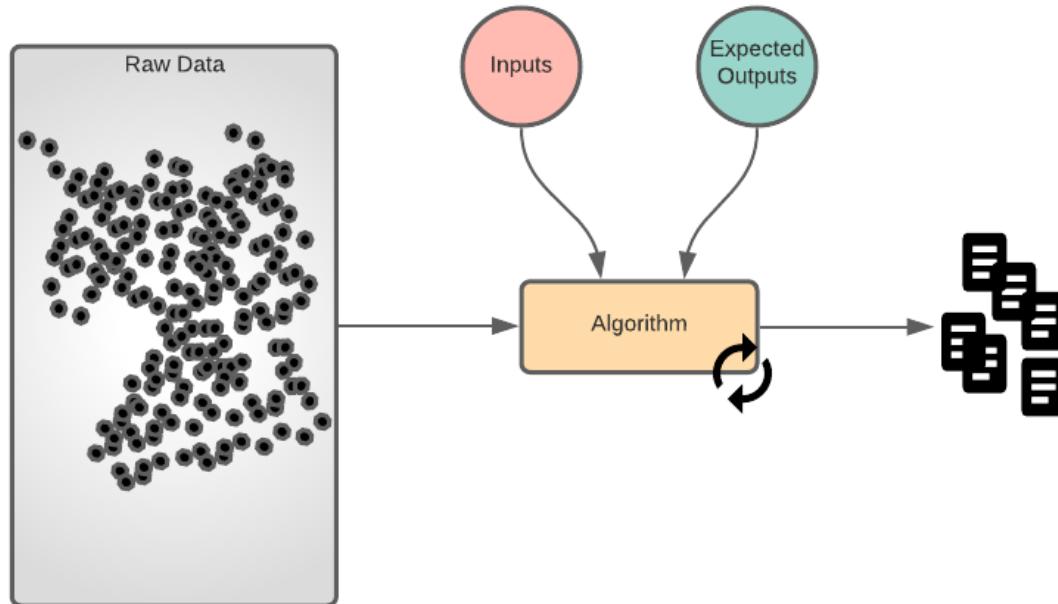
Machine Learning / Artificial Intelligence

- Machine learning algorithms are used to build mathematical models from existing data
- Results in a mathematical “trajectory” (and confidence level) for how new data will behave going forward
- The algorithms can be configured to learn and improve over time as more and more data is gathered and processed
- Three common approaches include:
 - Supervised learning
 - Unsupervised learning
 - Reinforcement learning

Supervised Learning

- The algorithm is provided with input data and expected output data
- The system learns by mapping and correlating the two
- The efficacy of the intelligence gained is dependent upon the accuracy of the inputs and outputs

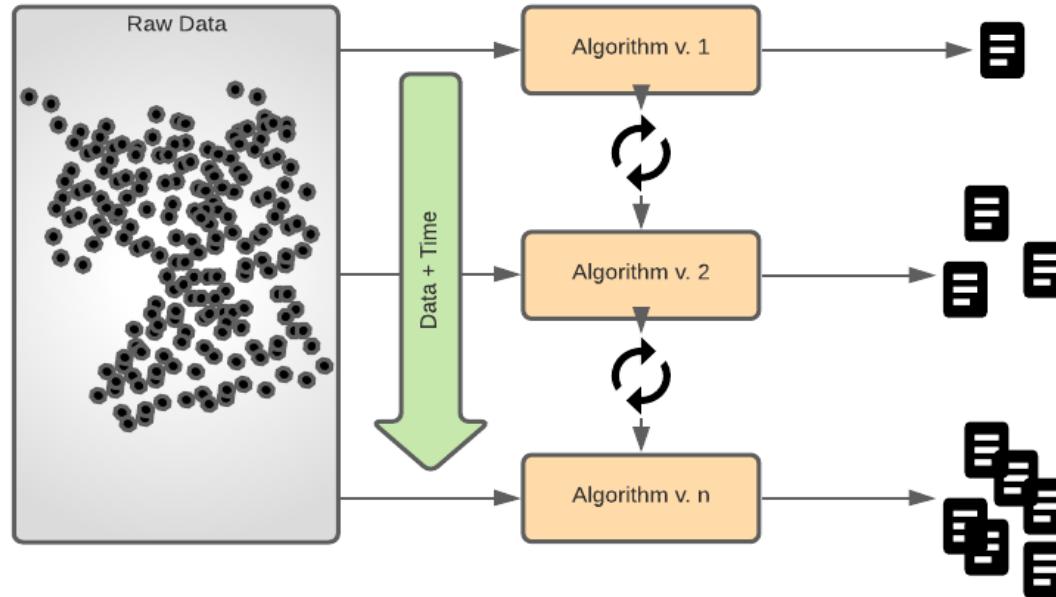
Supervised Learning



Unsupervised Learning

- The algorithm is given the data
- It uses Artificial Intelligence to dynamically discover and learn from patterns seen in the data
- The learning will likely be iterative – improving over time and with additional data volume

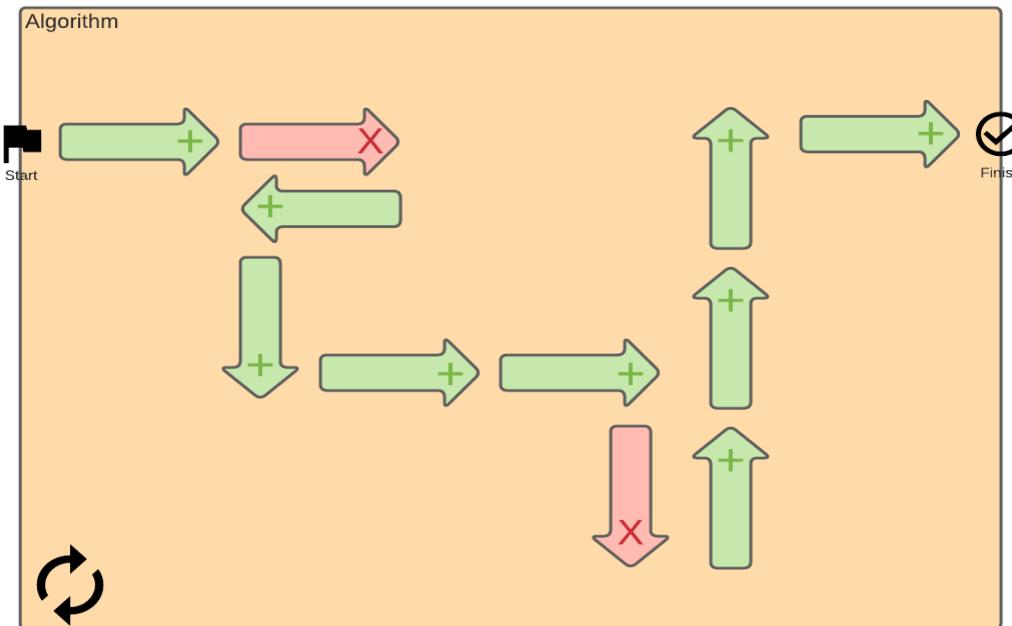
Unsupervised Learning



Reinforcement Learning

- The algorithm operates in an environment in which a sequence of steps is performed toward a specific goal
- Positive and negative ongoing feedback is provided as steps are executed
- The system attempts to minimize the negative and maximize the positive

Reinforcement Learning



The Turing Test

The **Turing test**, originally called the **imitation game** by Alan Turing in 1950,^[2] is a test of a machine's ability to exhibit intelligent behaviour equivalent to, or indistinguishable from, that of a human. Turing proposed that a human evaluator would judge natural language conversations between a human and a machine designed to generate human-like responses. The evaluator would be aware that one of the two partners in conversation is a machine, and all participants would be separated from one another. The conversation would be limited to a text-only channel such as a computer keyboard and screen so the result would not depend on the machine's ability to render words as speech.^[3] If the evaluator cannot reliably tell the machine from the human, the machine is said to have passed the test. The test results do not depend on the machine's ability to give correct **answers to questions**, only how closely its answers resemble those a human would give.

https://en.wikipedia.org/wiki/Turing_test

Machine Learning Examples

- See <https://www.businessinsider.com/shane-wighton-robotic-basketball-hoop-cant-miss-2020-5>
- See <https://breakingdefense.com/2020/08/ai-slays-top-f-16-pilot-in-darpa-dogfight-simulation/>
- See <https://www.schwab.com/automated-investing/what-is-a-robo-advisor>

Demo

Infrastructure-as-Code (IaC)

Azure Resource Manager (ARM) Templates

- In Azure, provides a JSON-based description language for defining infrastructure
- Supports parameterization, inputs, outputs, certain control structures
- Let's take a look...

AWS CloudFormation

- In AWS, provides either a YAML or JSON-based description language for defining infrastructure
- Supports parameterization, inputs, outputs, certain control structures
- Let's take a look...

Terraform

- While these components can be created manually in the MC, it's not ideal
- IaC (Infrastructure-as-Code) is a better alternative
- Infrastructure code (like all other code) is code
- Can be test, versioned, put in source control, etc.
- Multiple options available for tooling to support

What is Terraform?

- “infrastructure as code”
- *declarative* domain-specific language
 - what is declarative?
- used to describe *idempotent* resource configurations, typically in cloud infrastructure
- according to Hashicorp:
 - *Terraform enables you to safely and predictably create, change, and improve infrastructure. It is an open-source tool that codifies APIs into declarative configuration files that can be shared amongst team members, treated as code, edited, reviewed, and versioned*

What is Terraform? (cont'd)

- open source CLI tool for *infrastructure automation*
- utilizes plugin architecture
 - extensible to any environment, tool, or framework and works primarily by making API calls to those environments, tools, or frameworks
- detects implicit dependencies between resources and automatically creates a dependency graph
- builds in dependency order and automatically performs activities in parallel where possible
 - ...sequentially for dependent resources



Why Use Terraform?

- readable
- repeatable
- certainty (i.e., no confusion about what will happen)
- standardized environments
- provision quickly
- disaster recovery

What Does Terraform (HCL) Look Like?

```
resource "aws_instance" "web" {
    ami                  = "ami-
19827362728"
    instance_type = "t2.micro"

    tags = {
        Name = "my-first-instance"
    }
}
```

Hashicorp Configuration Language (HCL)

- The goal of HCL is to build a structured configuration language that is both human and machine friendly for use with command-line tools, but specifically targeted towards DevOps tools, servers, etc.
- Fully JSON compatible
- Made up of **stanzas** or **blocks**, which roughly equate to JSON objects. Each stanza/block maps to an object type as defined by **Terraform providers** (we'll talk more about providers later)
- <https://github.com/hashicorp/hcl>

Terraform Project Content Types

`*.tf, *.tf.json`

- HCL or JSON
- these files define your declarative infrastructure and resources

`*.tfstate`

- JSON files that store state, reference to resources
- created and maintained by terraform

`terraform.tfvars, terraform.tfvars.json` and/or `*.auto.tfvars, *.auto.tfvars.json`

- HCL or JSON
- variable definitions in bulk
- (more to come on setting variable values at runtime)

Resources

- *.tf files contain your **HCL declarative** definitions

```
resource "aws_instance" "web" {
    ami              = "ami-19827362728"
    instance_type   = "t2.micro"

    tags {
        Name = "my-first-instance"
    }
}
```

- most **blocks** in your HCL represent a **resource** to be created/maintained by Terraform

Resources

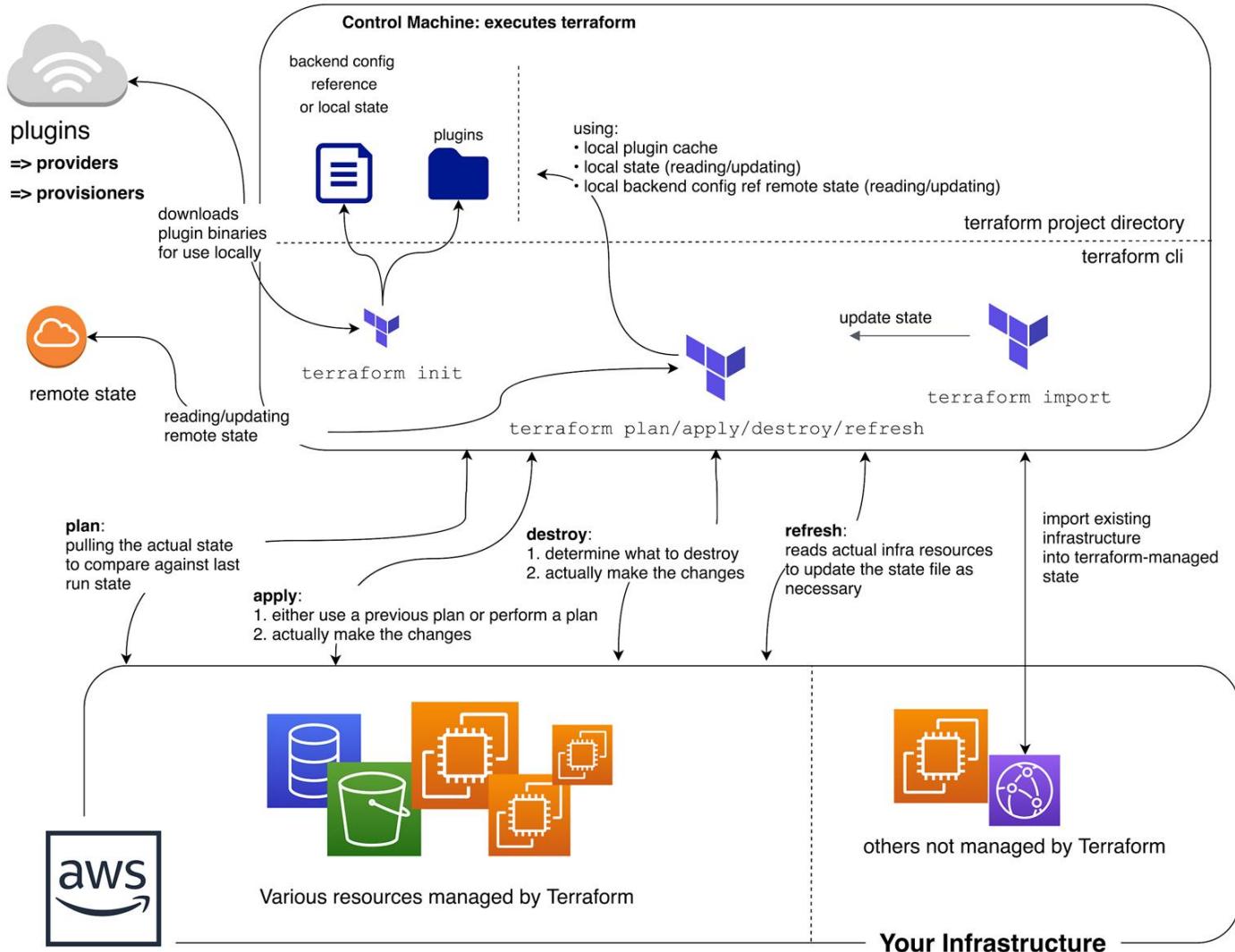
- resources are key elements and captured as top-level objects (stanzas) in Terraform configuration files
- each resource stanza indicates the intent to *idempotently* create that resource
- body of resource contains configuration of attributes of that resource
- each provider (e.g., AWS, Azure, etc.) provides its own set of resources and defines the configuration attributes
- when a resource is created by Terraform, it's tracked in Terraform *state*
- resources can refer to attributes of other resources, creating implicit dependencies
 - dependencies trigger sequential creation

Terraform Commands and the CLI

- The CLI is how you'll most often use terraform
 - `terraform init ...`
 - `terraform plan ...`
 - `terraform apply ...`
- And plenty more: `terraform --help` or <https://www.terraform.io/docs/commands/index.html>
- Third-party SDKs also available for running and interacting with Terraform (e.g., `scalr`, `terragrunt`, `terratest`)

Big picture look at

Terraform Command Flow



terraform init

- a special command, run before other commands/operations
- what does it do?
 - downloads required provider packages
 - downloads modules referenced in the HCL (more on modules later)
 - initializes state
 - local state: ensuring local state file(s) exist
 - remote state: more complex initialization (more on remote state later)
 - basic syntax check
- idempotent
- remember the `.terraform` directory?
 - **init** downloads the provider packages and modules to this directory
 - also, where state files live

Input Variables

- enable interchangeable values to be stored centrally and referenced single or multiple times
- similar to variables in other languages
- declared in **variable** stanzas
- parsed first
- cannot interpolate or reference other variables
- allow for default values
- optionally specify value type, e.g.,
 - **List**, **Map**, **String**

Input Variables

- Input variable definitions support the following
 - default – provides default value if not specified; makes optional
 - type – type of value accepted for the variable
 - description – string description/documentation
 - validation – block for defining validation rules for input
 - sensitive – true or false; limits output as part of TF operations (plan or apply)

Example Variable Definition

```
variable "instance_size" {
  default      = "t2.micro"
  type         = string # changed
  in 0.12
  description = "Size of EC2
  instance"
}
```

Example Variable Definition

```
variable "student_alias" {
  type          = string
  description = "Your student alias"
  validation {
    condition      = trimprefix(var.student_alias, "test") ==
var.student_alias
    error_message = "Please do not use test aliases with this
deployment."
  }
}
```

Data Sources

- logical references to data objects stored externally to the **tfstate** file
- allows you to reference resources not created by Terraform
- examples
 - current default region in AWS CLI
 - AMI ID search
 - AWS ARN lookup
 - AWS VPC CIDR range

Data Source Example: AWS AMI Lookup

```
data "aws_ami" "latest-ubuntu" {
  most_recent = true
  owners      = ["099720109477"]

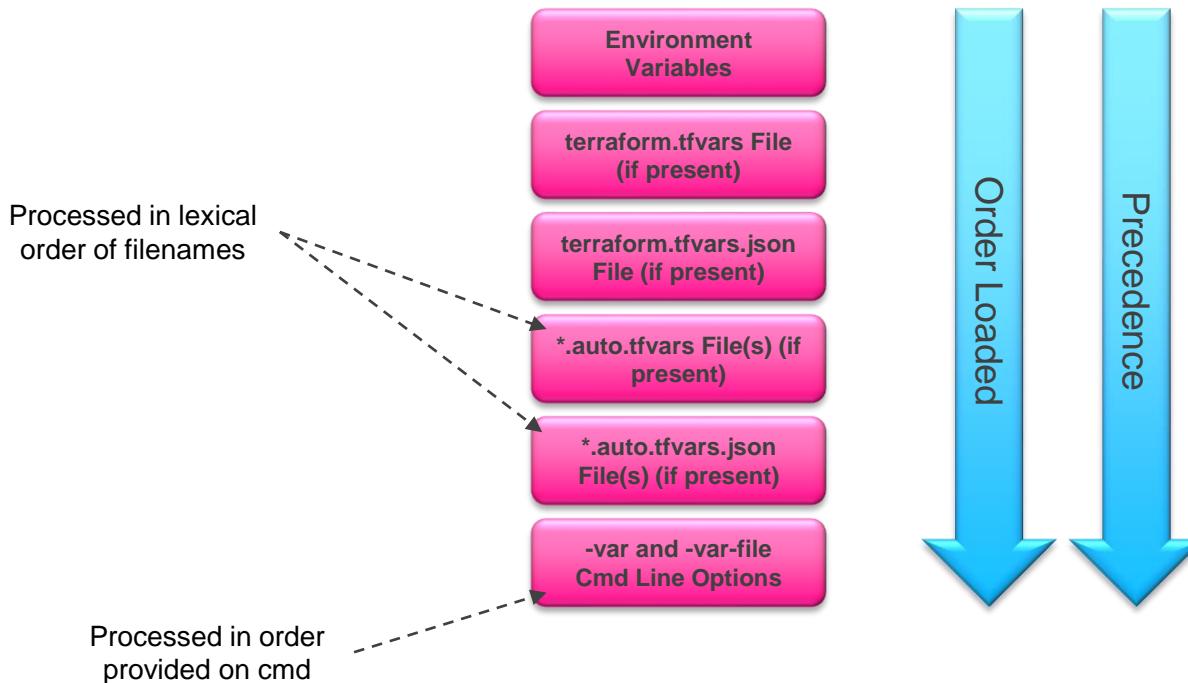
  filter {
    name    = "name"
    values  = ["ubuntu/images/hvm-ssd/ubuntu-xenial-16.04-amd64-
server-*"]
  }

  filter {
    name    = "virtualization-type"
    values  = ["hvm"]
  }
}
```

Providing Values for Input Variables

- Multiple options
 - using environment variables (prefixed with “TF_VAR_”)
 - defining inputs in a `terraform.tfvars` file
 - defining inputs in a `terraform.tfvars.json` file
 - defining inputs in one or more `*.auto.tfvars` files
 - defining inputs in one or more `*.auto.tfvars.json` files
 - `-var` and `-var-file` options on the command-line

Providing Values for Input Variables



Providing Values for Input Variables

- primarily used when executing Terraform via CLI
- not really used with Terraform Enterprise
- can “push” those variables + values to Enterprise (in files)
- but manage from “Variables” section of the environment

State

- stores information about resources that are created by Terraform
 - also includes values computed by the provider APIs
- local file
 - **.tfstate**
- or backends are also available...

Backends

- determines how state is loaded and how operations like `apply` are executed
- enables non-local file state storage, remote execution, etc.
- why use a backend?
 - can store their state remotely and protect it to prevent corruption
 - some backends, e.g., *Terraform Cloud* automatically store all revisions
 - keep sensitive information off local disk
 - remote operations
 - `apply` can take a *LONG* time for large infrastructures

Backends (cont'd)

- examples
 - S3
 - swift
 - http
 - Terraform Enterprise
 - etc.

Providers

- responsible for understanding API interactions and exposing resources
- Hashicorp helps companies create providers to be added to ecosystem
- declared in HCL config files as a **provider** stanza
- each Terraform project can have multiple providers, even of the same type
- describes resources, their inputs, outputs, and the logic to create and change them
- many options
 - AWS, GCP, Azure, and many others
 - providers available for non-infra services as well such as gmail, MySQL, and Pagerduty

The AWS Provider

- provider documentation
 - <https://www.terraform.io/docs/providers/aws/index.html>
- HUGE amount of resources
- something like 8 resources per service on average

Configuring the Provider

```
provider "aws" {
    region      = "us-west-1"
    access_key = "[your access key]"
    secret_key = "[your secret access
key]"
}
```

Output Variables

- *inputs* to a Terraform config are declared with variables stanzas
- *outputs* are declared with a special output stanza
- can be referenced through the modules interface or the CLI

Output Variables

- Output variable definitions support the following
 - value – value to be returned as output
 - description – string description/documentation
 - sensitive – true or false; limits output as part of TF operations (plan or apply)

Output Definition

```
output "instance_public_ip" {
    value = aws_instance.web.public_ip
}
```

Demo

Docker

Setting Up a Local Docker Dev Environment

Tools:

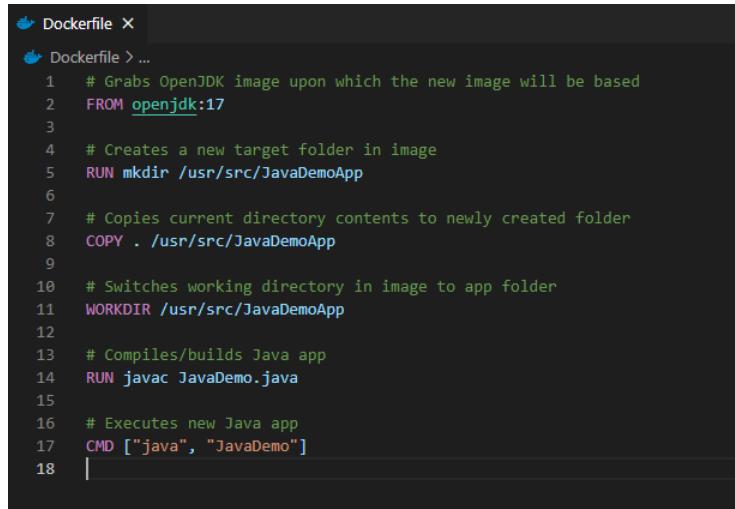
- Docker Desktop
- Code editor/IDE that plays nice with containers (e.g., VS Code)

The Dockerfile

- Tells Docker what to do in creating an image for your application
- The commands are all things you could do from the CLI
- Used by the docker “build” command
- Docker build uses this file and a “context” – a set of files at a specified location – to make your image

Dockerfile example

- The following creates an image for building/running Java app in container
- See <https://github.com/KernelGamut32/dockerlab-repo-sample> for sample



A screenshot of a code editor displaying a Dockerfile. The file contains 18 numbered lines of Docker build instructions. The code uses color-coded syntax highlighting for commands like FROM, RUN, COPY, WORKDIR, and CMD.

```
1  # Grabs OpenJDK image upon which the new image will be based
2  FROM openjdk:17
3
4  # Creates a new target folder in image
5  RUN mkdir /usr/src/JavaDemoApp
6
7  # Copies current directory contents to newly created folder
8  COPY . /usr/src/JavaDemoApp
9
10 # Switches working directory in image to app folder
11 WORKDIR /usr/src/JavaDemoApp
12
13 # Compiles/builds Java app
14 RUN javac JavaDemo.java
15
16 # Executes new Java app
17 CMD ["java", "JavaDemo"]
18 |
```

Docker Images

- Represent templates defining an application environment
- New instances of the application can be created from the image
- These instances are called containers

Docker Images

- Images are defined via a Dockerfile definition
- Support layers for building up the environment in stages
- Fully defines the application, including all components required to support

Docker Images

Those components can include:

- Runtime
- Development framework
- Source code
- Executable instructions for container startup

Docker Images

Start with a base that gives your app a place to live
Needed OS/runtimes/dB server applications, etc.

Examples:

nginx

Node

MySQL

Apache HTTP Server

IIS with .NET Runtimes

Docker Hub

- Centralized registry for image storage & sharing
- Can signup for an account – user accounts offer both free and pro versions
- Also, supports organizations for grouping of multiple team members

Docker Hub

- Accessible at <https://hub.docker.com>
- Search feature enables search for image by technology or keyword
- Image detail displays available tags and image variants

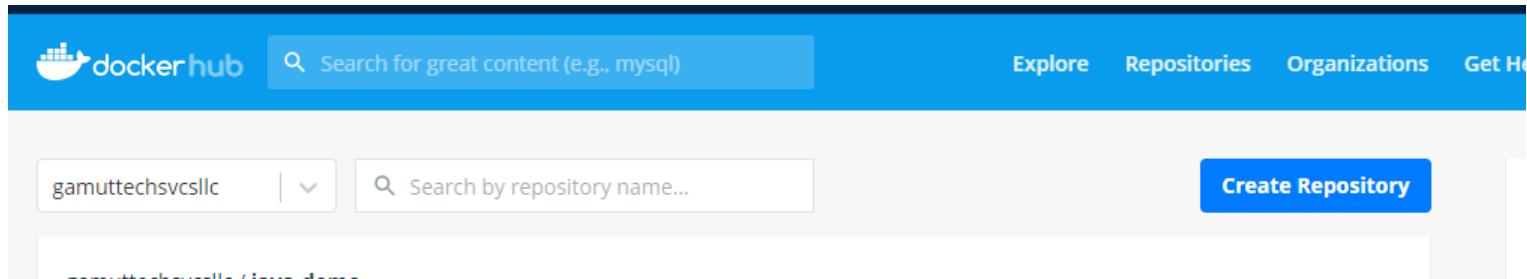
Docker Hub

- May also include examples of usage
- Pro account supports image scanning for security vulnerabilities
- Can be useful for image reuse and image sharing across a dev team

Docker Hub

There are other registry types, including private registries

Docker Hub



Docker Hub

The screenshot shows the Docker Hub interface. At the top, there's a search bar with the query "ruby". Below the search bar, there are tabs for "Explore", "Repositories", and "Organizations". A prominent blue button labeled "Create Repos" is visible. On the left, a sidebar lists several user profiles under "Verified Content (4)": "gamuttechsvcsllc", "jruby", "gamuttechsvcsllc / ja", "redmine", "rails", "gamuttechsvcsllc / ss", "Community (16203)", "rubylang/ruby", "rubylang/cf-uaac", "rubylang/rubyfarm", "rubylang/fog-google", and "gamuttechsvcsllc / e". Each item has a timestamp indicating when it was last updated. The main content area displays four repository cards for "ja", "ss", "cf-uaac", and "e". Each card includes a status icon ("Not Scanned"), star count (0), download count (4, 40, 28, 22), and a "Private" or "Public" access level indicator.

Repository	Status	Stars	Downloads	Access
ja	Not Scanned	0	4	Private
ss	Not Scanned	0	40	Public
cf-uaac	Not Scanned	0	28	Public
e	Not Scanned	0	22	Public

Docker Hub

The screenshot shows the Docker Hub interface. At the top, there's a blue header bar with the Docker Hub logo, a search bar containing 'ruby', and navigation links for 'Explore', 'Repositories', 'Organizations', 'Get Help', and a user profile for 'gamtuttechsvcslic'. Below the header, there are three tabs: 'Docker', 'Containers' (which is selected), and 'Plugins'. On the left, there are 'Filters' for 'Images' (with options for 'Verified Publisher' and 'Official Images') and 'Categories' (with options for 'Analytics' and 'Application Frameworks'). The main search results area displays 1 - 25 of 16,207 results for 'ruby'. A 'Most Popular' dropdown menu is open. The first result is for the 'Ruby' official image, which has a red icon, was updated a day ago, and is described as a dynamic, reflective, object-oriented, general-purpose, open-source programming language. It supports various architectures like Container, Linux, 386, x86-64, ARM 64, IBM Z, mips64le, ARM, PowerPC 64 LE, and Programming Languages. It has over 10M+ downloads and 2.0K stars. There's also another 'OFFICIAL IMAGE' button below it.

Docker Hub



ruby ☆

Docker Official Images

Ruby is a dynamic, reflective, object-oriented, general-purpose, open-source programming language.

↓ 100M+

Container Linux PowerPC 64 LE 386 x86-64 ARM 64 IBM Z mips64le ARM Programming Languages

Official Image

Copy and paste to pull this image

`docker pull ruby`



[View Available Tags](#)

Description

Reviews

Tags

Docker Hub

How to use this image

Create a `Dockerfile` in your Ruby app project

```
FROM ruby:2.5

# throw errors if Gemfile has been modified since Gemfile.lock
RUN bundle config --global frozen 1

WORKDIR /usr/src/app

COPY Gemfile Gemfile.lock .
RUN bundle install

COPY .

CMD ["./your-daemon-or-script.rb"]
```

Put this file in the root of your app, next to the `Gemfile`.

You can then build and run the Ruby image:

```
$ docker build -t my-ruby-app .
$ docker run -it --name my-running-script my-ruby-app
```

Generate a `Gemfile.lock`

Building The Image

- To build the image from Dockerfile use *docker build*
- *docker build -t <tag name> <path to Dockerfile>*
- For example, *docker build -t java-demo .*
- Builds image from Dockerfile in current folder(.) with tag name “java-demo”

Building The Image

- For tag name, can include optional detail:
 - Docker ID in Docker Hub for eventual push to image registry
 - Version identifier for tag – defaults to “latest” if excluded
- For example, *docker build -t <docker ID>/<tag name>:<version> .*

Application Bootstrapping with Docker and k8s

- Kubernetes provides a hosting environment for containerized applications
- Once you have a Docker image, you can work entirely within Kubernetes to deploy your app

Open Container Initiative (OCI)

- The OCI is an open governance structure for the express purpose of creating open industry standards around container formats and runtimes
- Docker started it
- They have two specs: runtime-spec and image-spec
- This deals with containers in the abstract

Competing Container Runtimes

[rkt](#) from CoreOS

[Mesos](#) from Apache

[LXC](#) Linux containers

Kubernetes and Container Orchestration

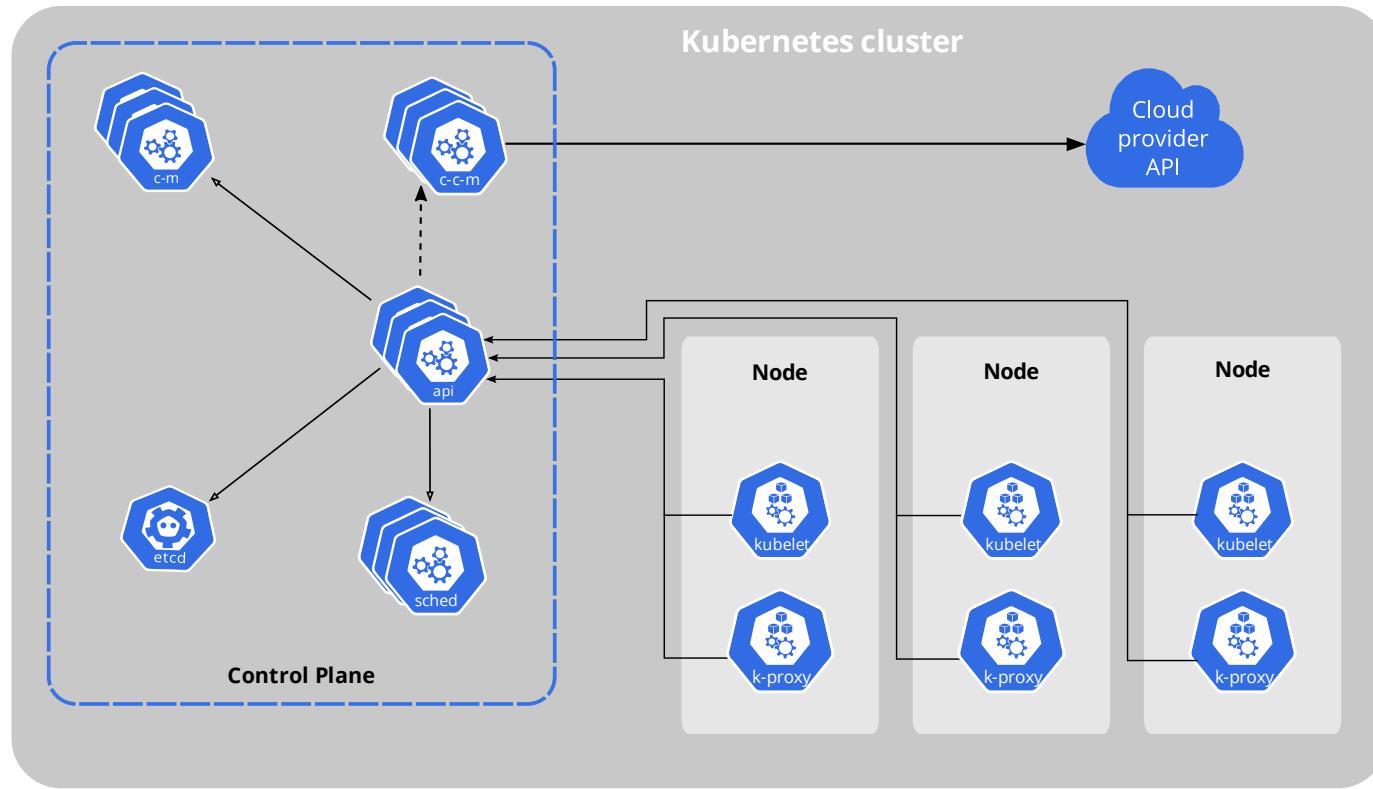
Kubernetes (k8s) Overview

- Kubernetes is a portable, extensible, open-source platform for managing containerized workloads and services

What is an Orchestrator and Why Do We Need It?

- What would it look like to manually control the containers needed for your application as your app scales or containers fail?
- “Orchestration” is the execution of a defined workflow
- We can use an orchestrator to start and stop containers automatically based on your set rules
- What does this open up for you?

Architecture of k8s System



Core Components of k8s

- When you deploy Kubernetes, you get a cluster.
- A Kubernetes cluster consists of a set of worker machines, called nodes, that run containerized applications. Every cluster has at least one worker node.
- The worker node(s) host the Pods that are the components of the application workload.
- The control plane manages the worker nodes and the Pods in the cluster.

k8s Resource/Manifest

- Kubernetes manifests are text files, usually written in YAML
- They are used to create, modify and delete Kubernetes resources

Kubernetes Architecture

- Kubernetes automates and monitors the lifecycle of a stateless application
- It can scale the app up or down and ensure it keeps running
- Kubernetes cluster consists of computers called nodes
- The basic unit of work in kubernetes is called a pod, which is a group of one or more containers
- Kubernetes can be divided into planes
 - Control plane - the actual pods that the kubernetes core components runs on, exposing the api, etc...
 - Data Plane - everything else meaning the nodes and pods which the application runs on

Kubernetes Architecture

- In short, control plane is what monitors the cluster, schedules the work, makes the changes, etc...
- The nodes are what actually do the real work, report back to the master, and watch for changes

Kubernetes Control Plane & Data Plane

- Kubernetes is actually a collection of pods implementing the k8s api and the cluster orchestration logic (AKA the Kubernetes control plane)
- The application/data plane is everything else, meaning all the nodes that host all the pods that the application runs on
- The *controllers* of the control plane implement control loops that repeatedly compare the desired state of the cluster to its actual state
- When the state of the cluster changes, controllers take action to bring it back inline with desired state

Declarative and Desired State

- Kubernetes supports a declarative model – we can send the api a yaml file in a declarative form
- Desired state means that we specify in the yaml file our desired state and the cluster takes responsibility to make sure it will happen
- We describe the desired state using a yaml or json file that serves as a record of intent, but we do not specify how to get there (this is kubernetes responsibility to get us there)
- Things could change or go wrong over the lifetime of the cluster (node failing, etc...), Kubernetes is responsible to always make sure that the desired state is kept intact
- Kubernetes control plane controllers are always running in a loop and checking that the actual state of the cluster matches the desired state, so that if any error occurs they kick in and rectify the cluster

Reconciling state

- Watch for the **spec** fields in the YAML files
- The **spec** describes *what we want the thing to be*
- Kubernetes will *reconcile* the current state with the spec (technically, this is done by a number of *controllers*)
- When we want to change a resource, we update the **spec**, and reapply
- Kubernetes will then *converge* that resource

k8s Pods: The Basic Building Block

- A pod represents a set of running containers in your cluster
- Worker nodes host pods
- The control plane manages the worker nodes and the pods inside them

k8s Pods: The Basic Building Block

- Pod is the basic execution and scaling unit in Kubernetes
- Kubernetes runs containers but always inside pods
- It is like a sandbox in which containers run (abstraction)
- A pod is a group of containers:
 - Running together (on the same node)
 - Sharing resources (RAM, CPU; but, also, network, volumes, etc...)
- A Pod models an application-specific “logical host”

Pod state

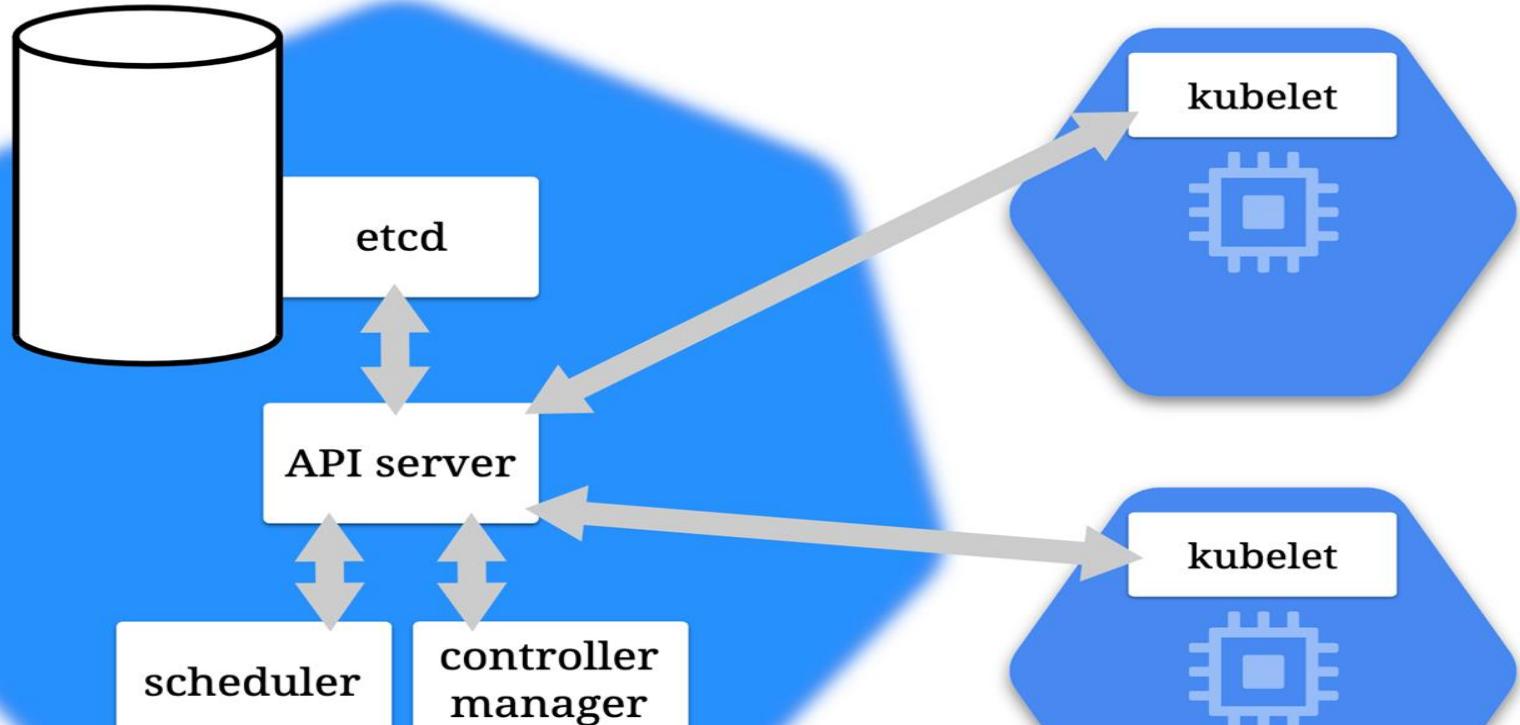
- Pods are considered to be relatively ephemeral (rather than durable) entities
- Pods do not hold state, so if a pod crashes, Kubernetes will replace it with another
- Multiple instances of the same pod are called replicas

Deployments

- A *Deployment* controller provides declarative updates for Pods and ReplicaSets
- The Deployment Object gives us a better way of handling the scaling of pods
- The advantage of using Deployment versus using a replicaset is having rolling updates support for the pod container versions out-of-the-box

Deployments

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

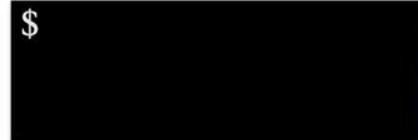


CONTROL PLANE

WORKER NODES



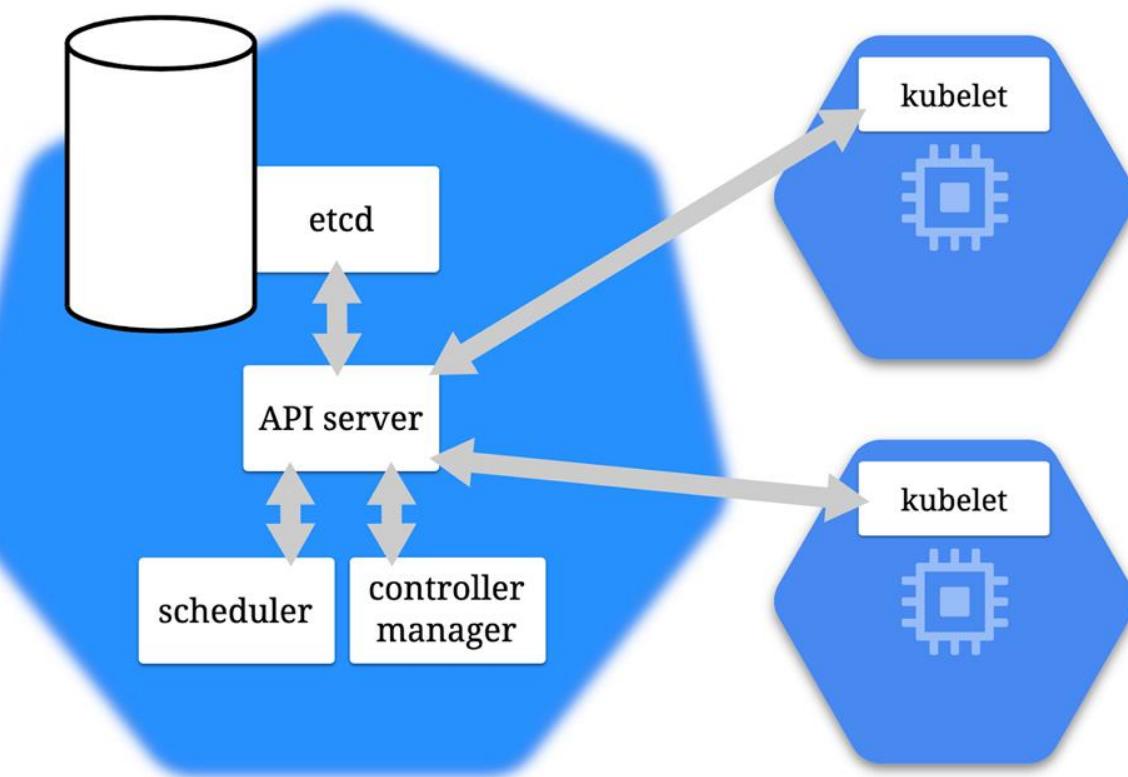
\$



DEVOPS

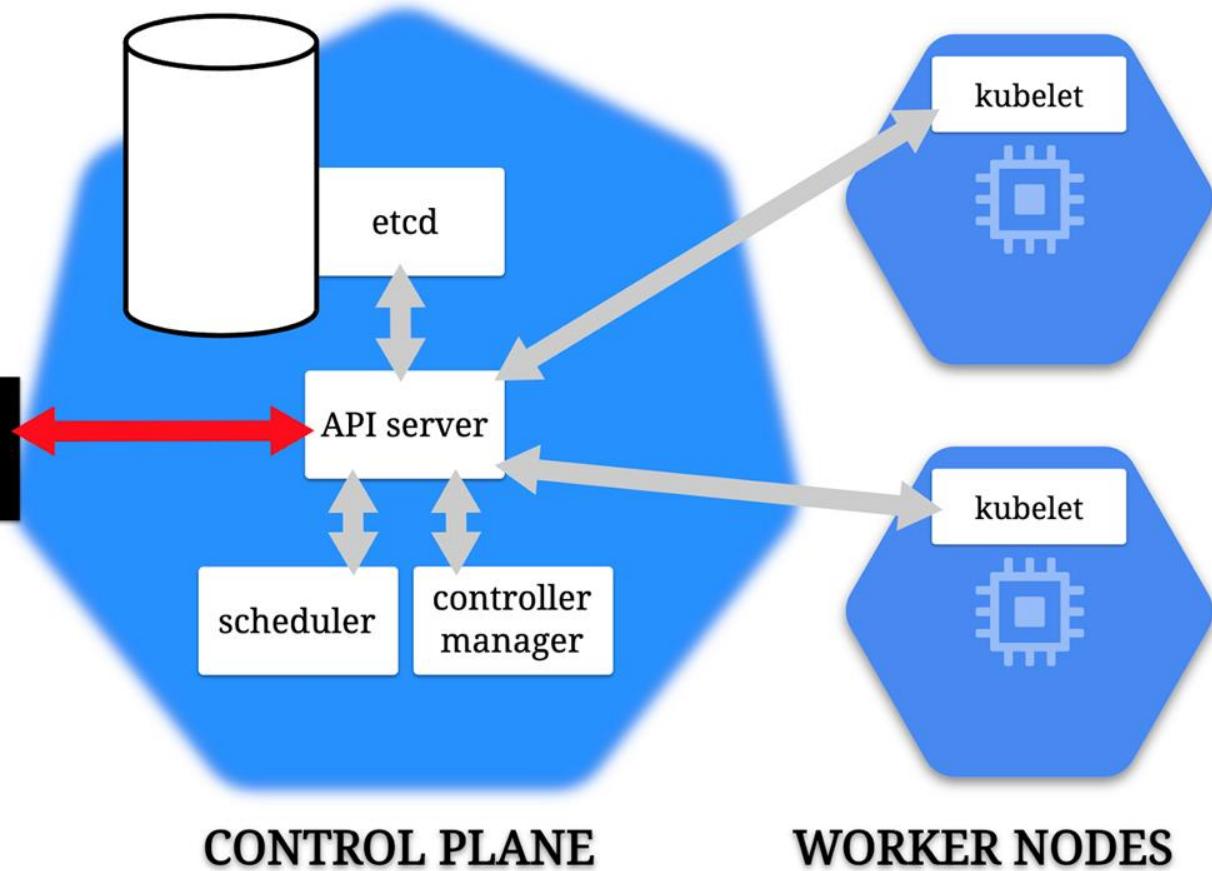
CONTROL PLANE

WORKER NODES





```
$ kubectl run web \  
--image=nginx \  
--replicas=3
```



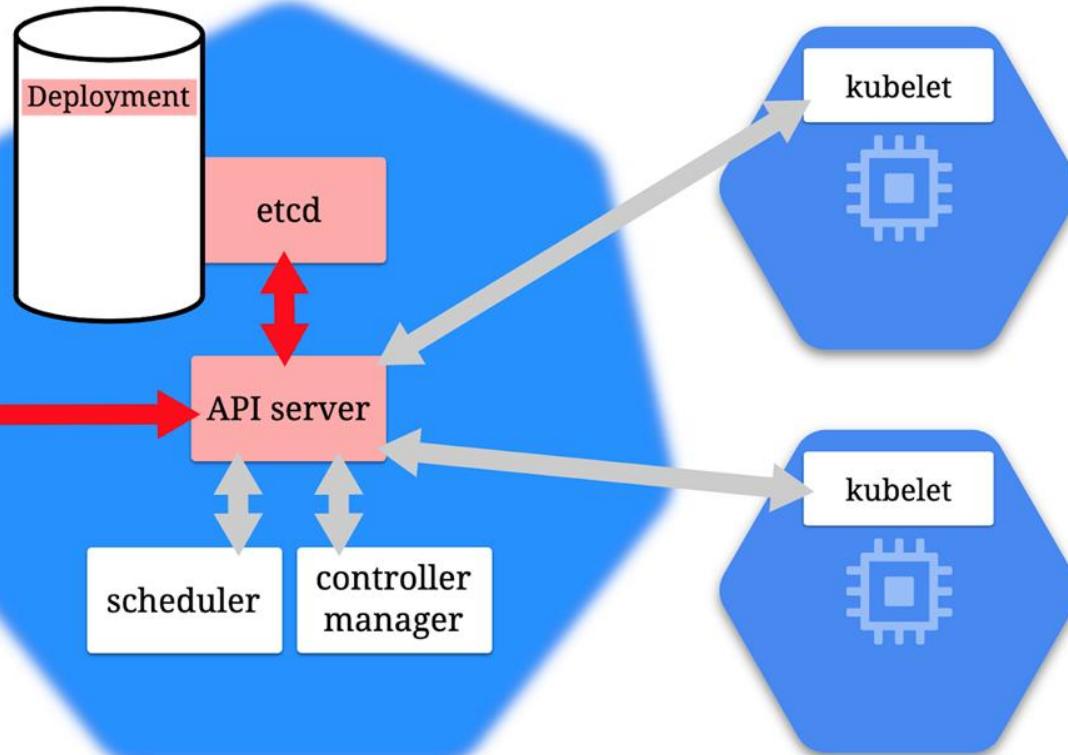
DEVOPS

CONTROL PLANE

WORKER NODES



```
$ kubectl run web \  
--image=nginx \  
--replicas=3
```



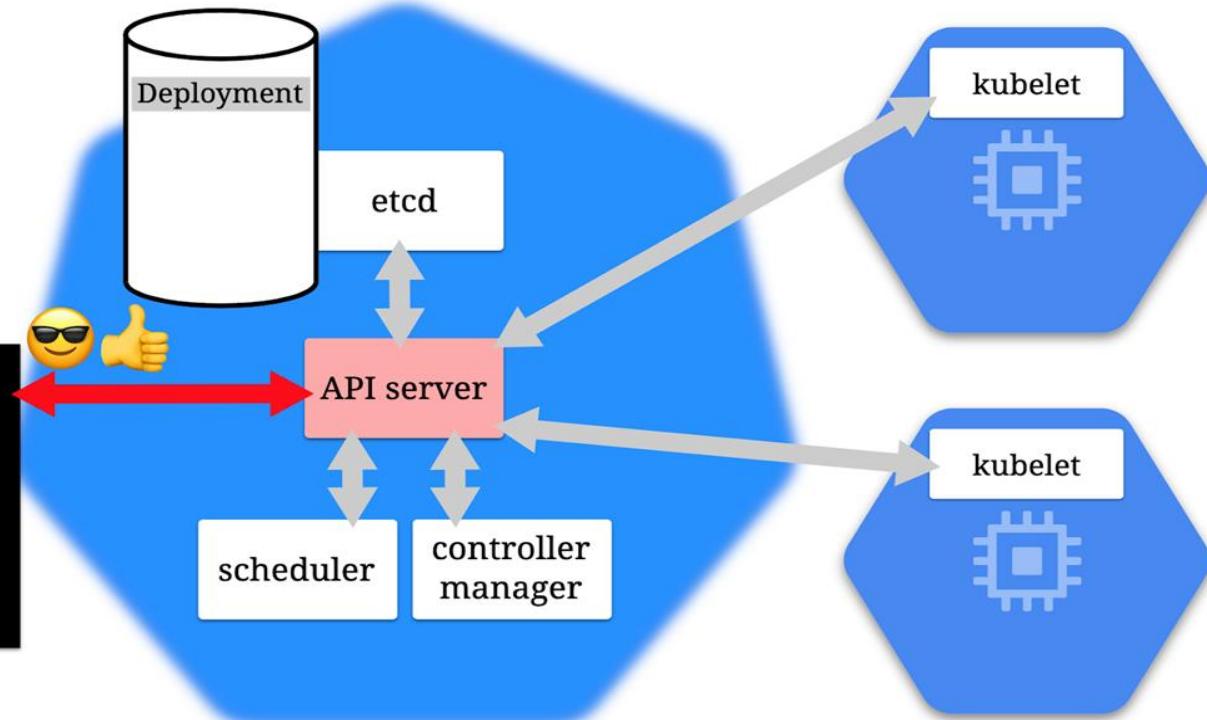
DEVOPS

CONTROL PLANE

WORKER NODES



```
$ kubectl run web \
--image=nginx \
--replicas=3
...
deployment.apps/web
created
$
```



DEVOPS

CONTROL PLANE

WORKER NODES

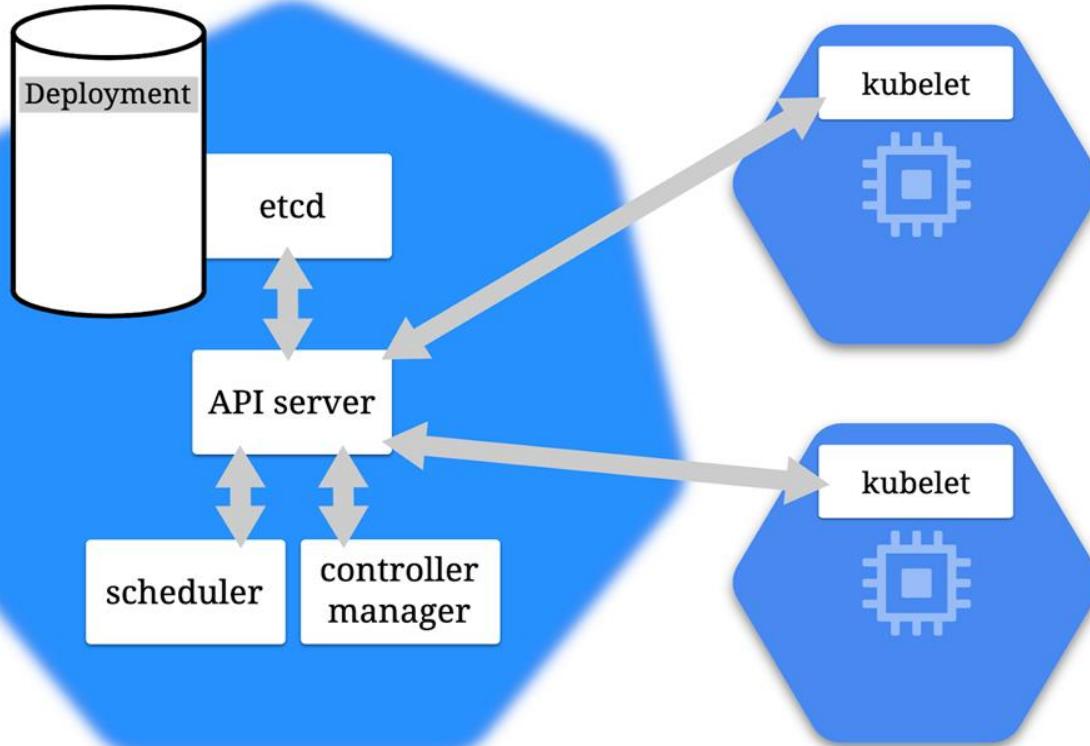


\$

DEVOPS

CONTROL PLANE

WORKER NODES





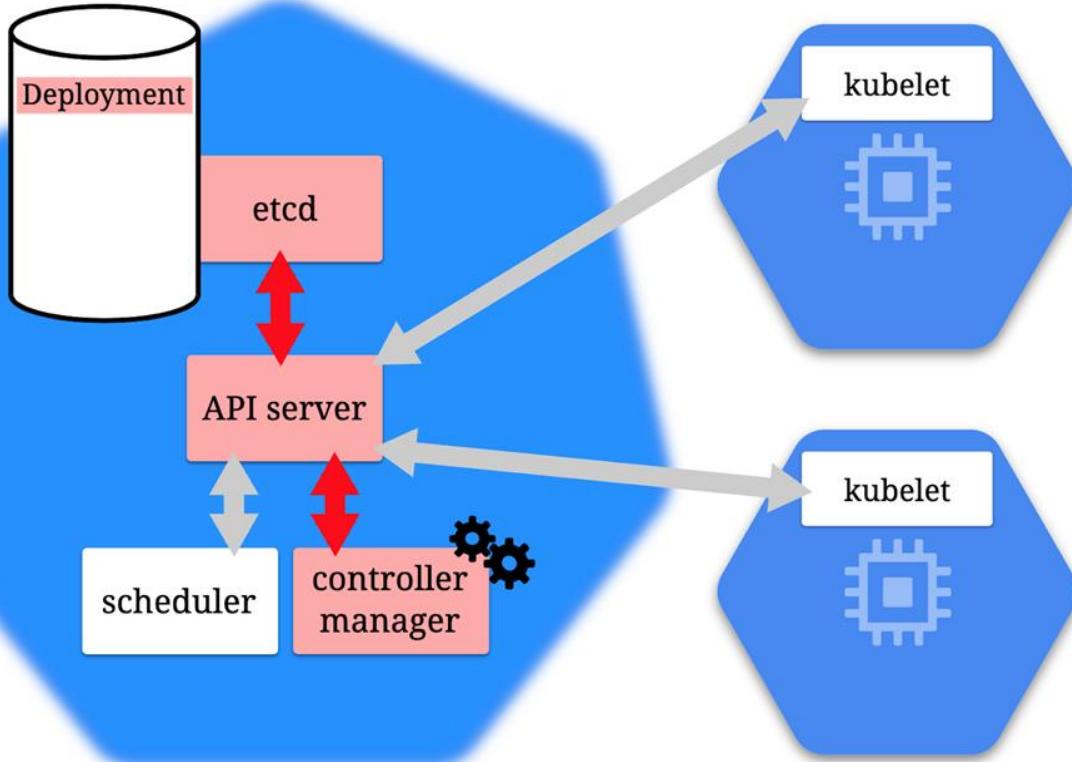
\$



DEVOPS

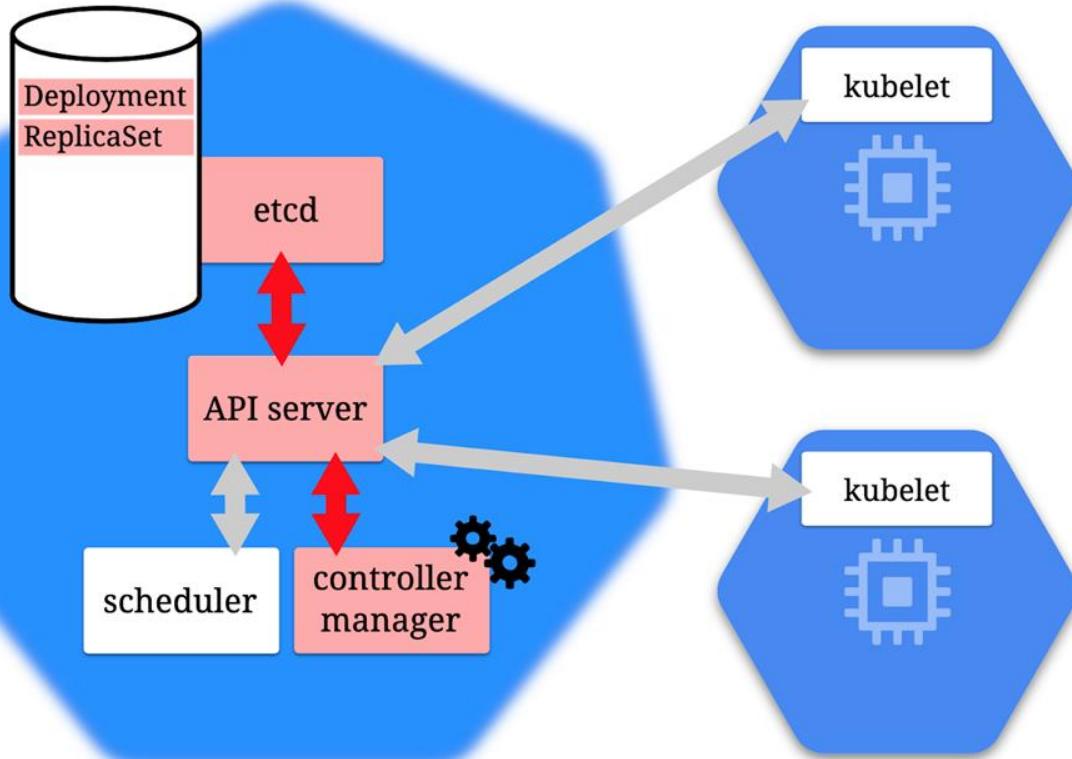
CONTROL PLANE

WORKER NODES





\$

DEVOPS**CONTROL PLANE****WORKER NODES**

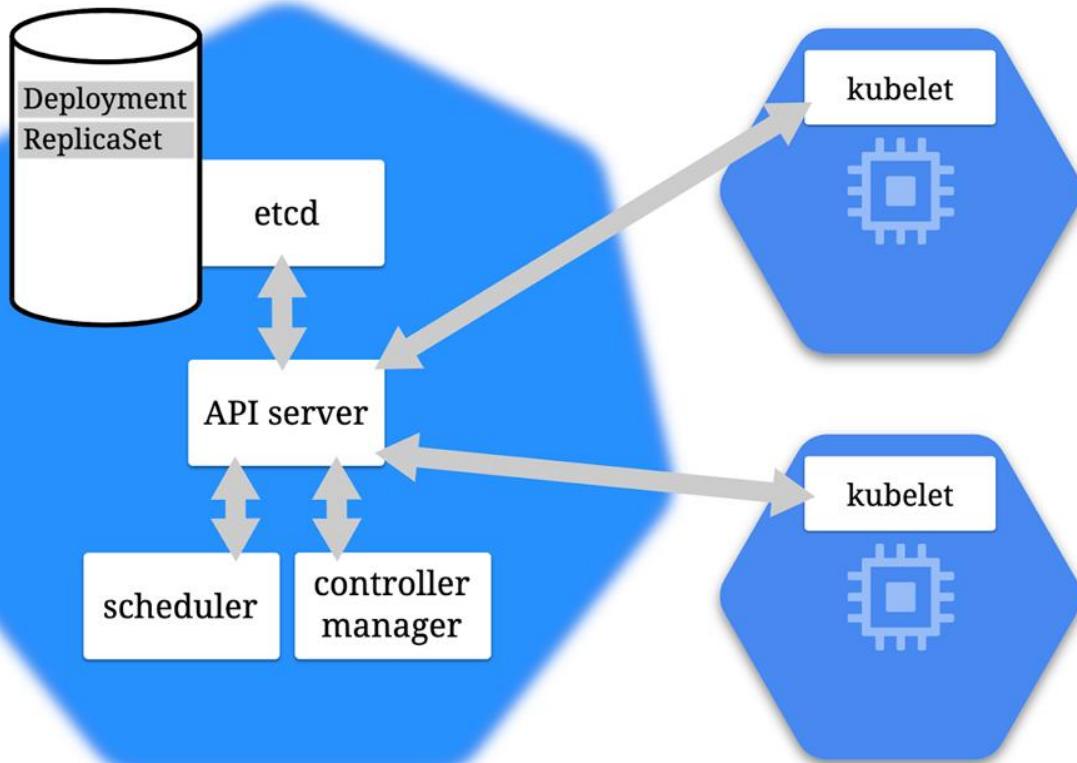


\$ [REDACTED]

DEVOPS

CONTROL PLANE

WORKER NODES



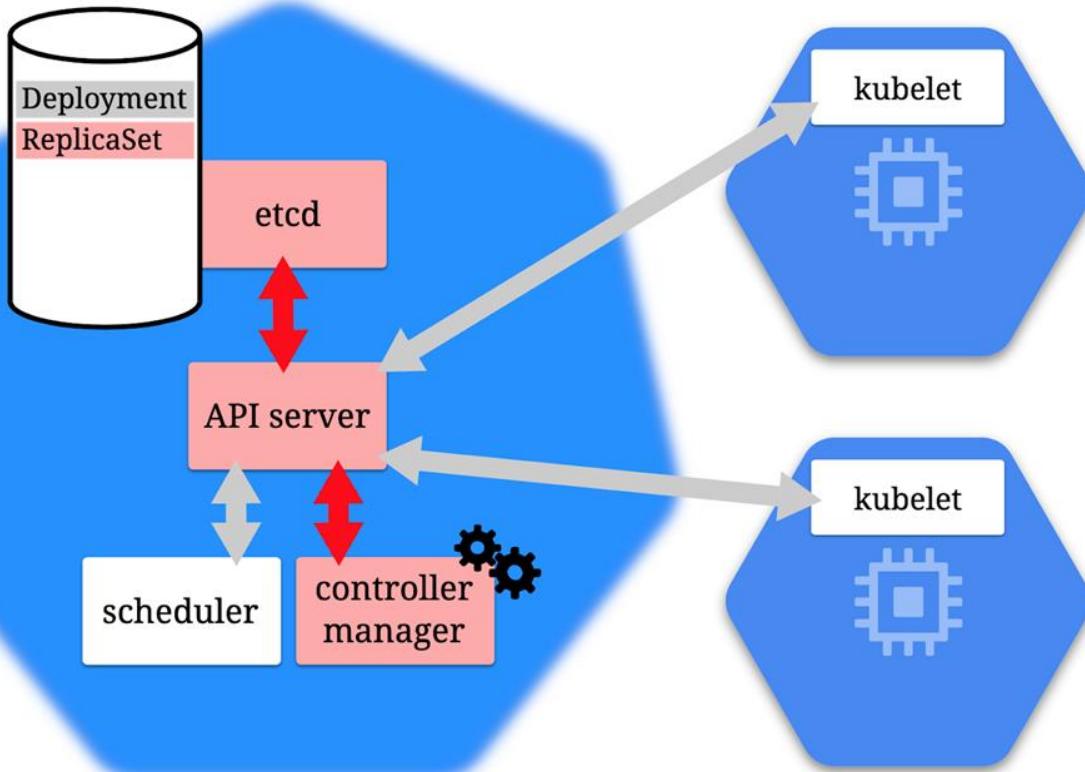


\$ [REDACTED]

DEVOPS

CONTROL PLANE

WORKER NODES



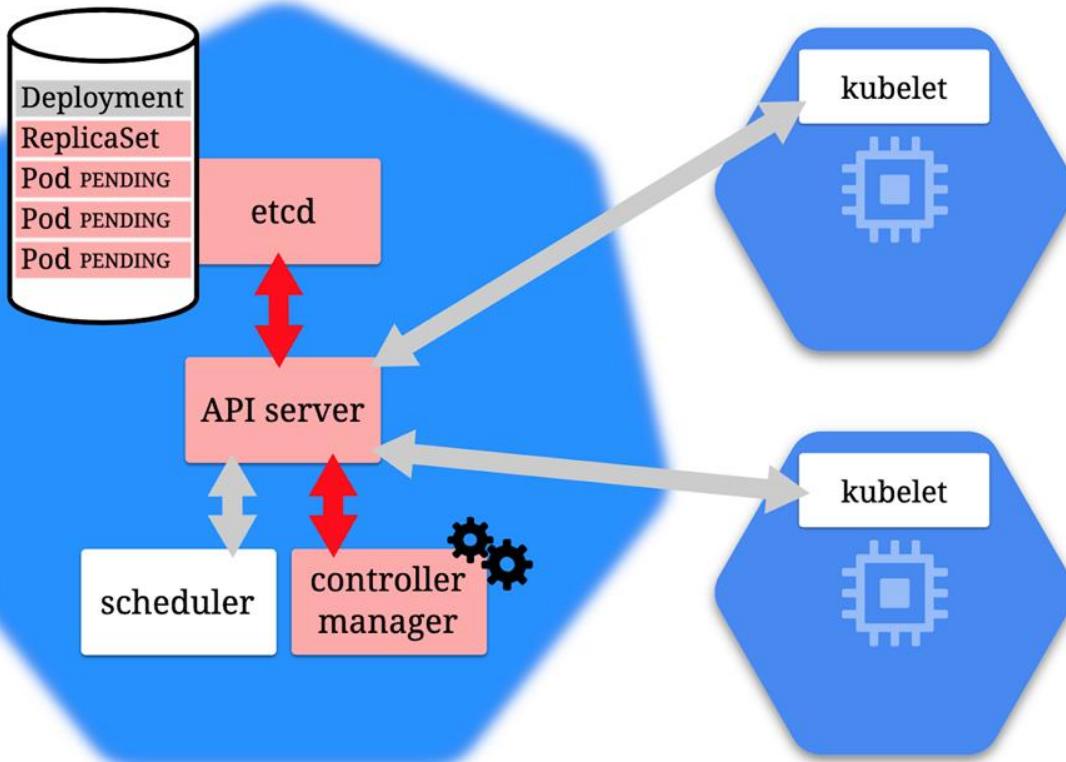


\$ [REDACTED]

DEVOPS

CONTROL PLANE

WORKER NODES



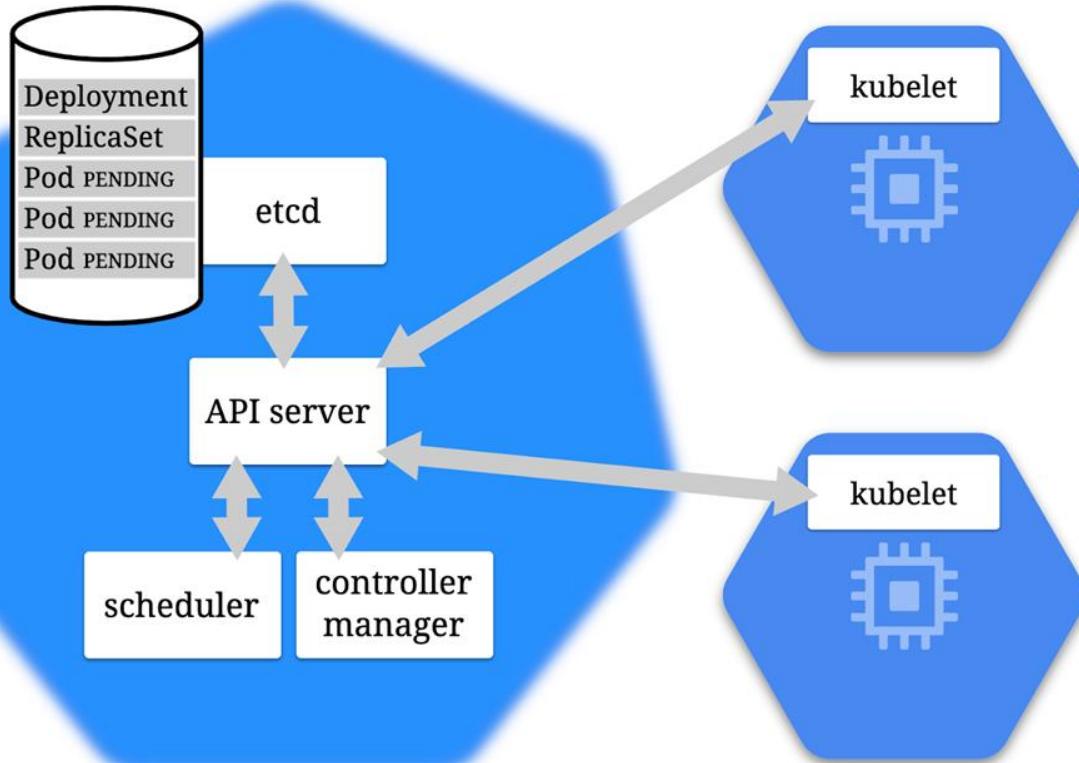


\$ [REDACTED]

DEVOPS

CONTROL PLANE

WORKER NODES

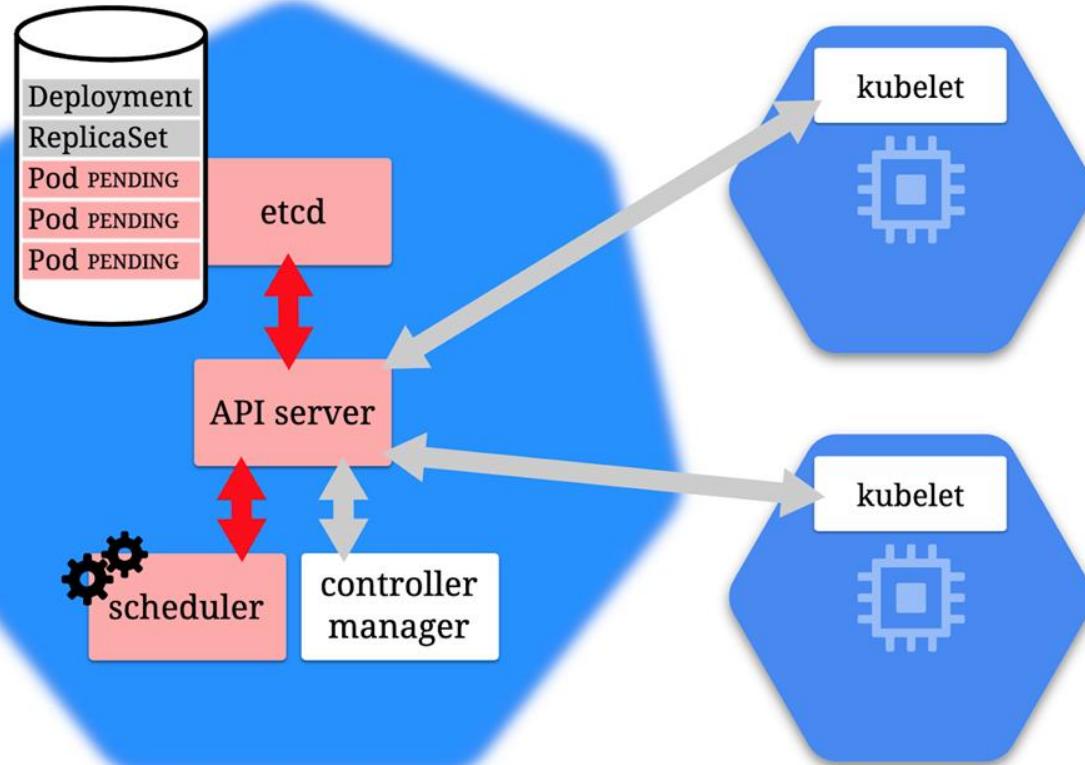




DEVOPS

CONTROL PLANE

WORKER NODES



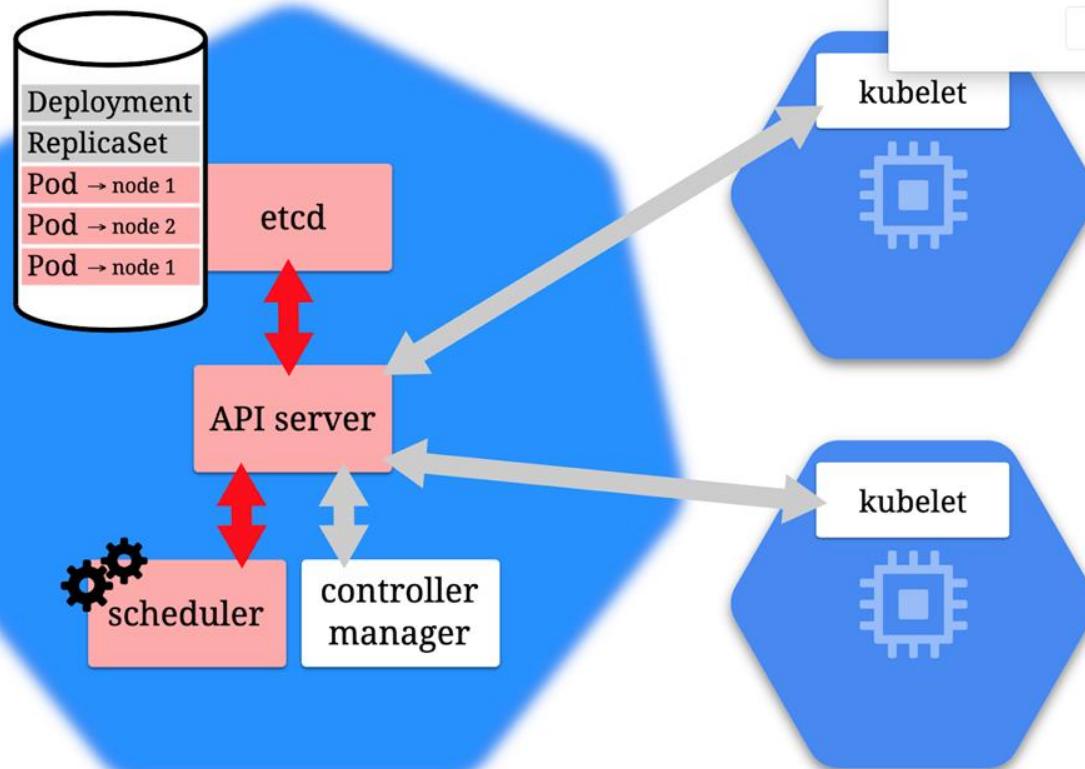


\$

DEVOPS

CONTROL PLANE

WORKER NODES



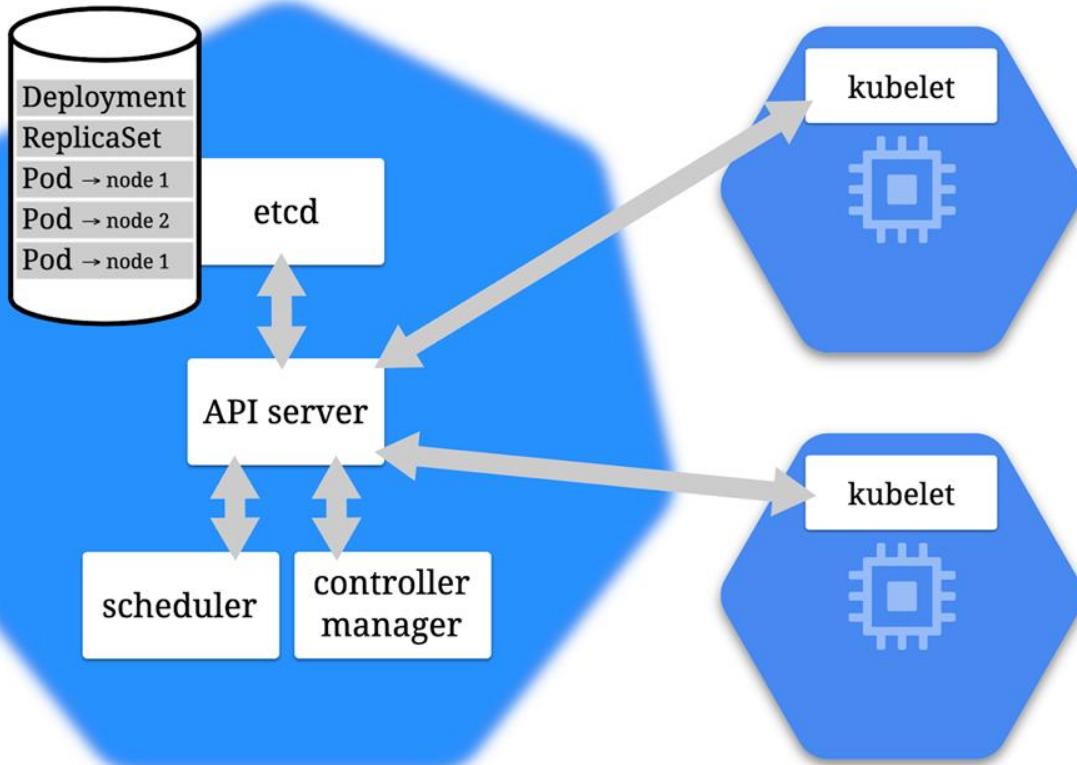


\$

DEVOPS

CONTROL PLANE

WORKER NODES



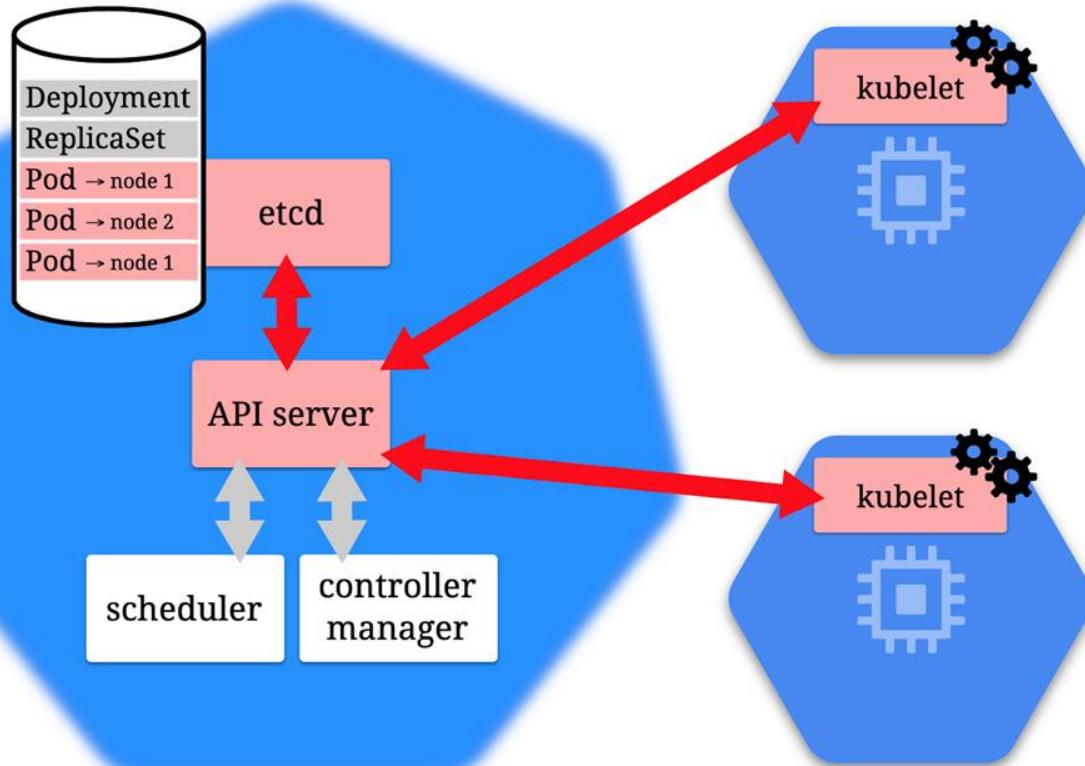


\$

DEVOPS

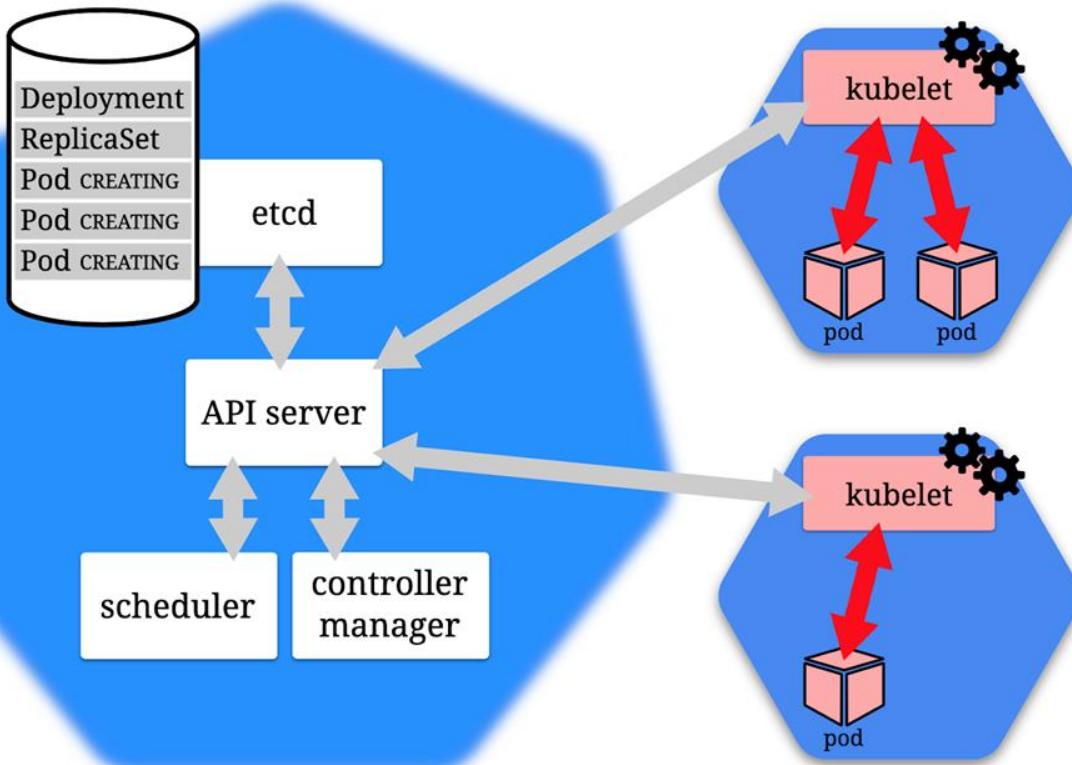
CONTROL PLANE

WORKER NODES





DEVOPS



CONTROL PLANE

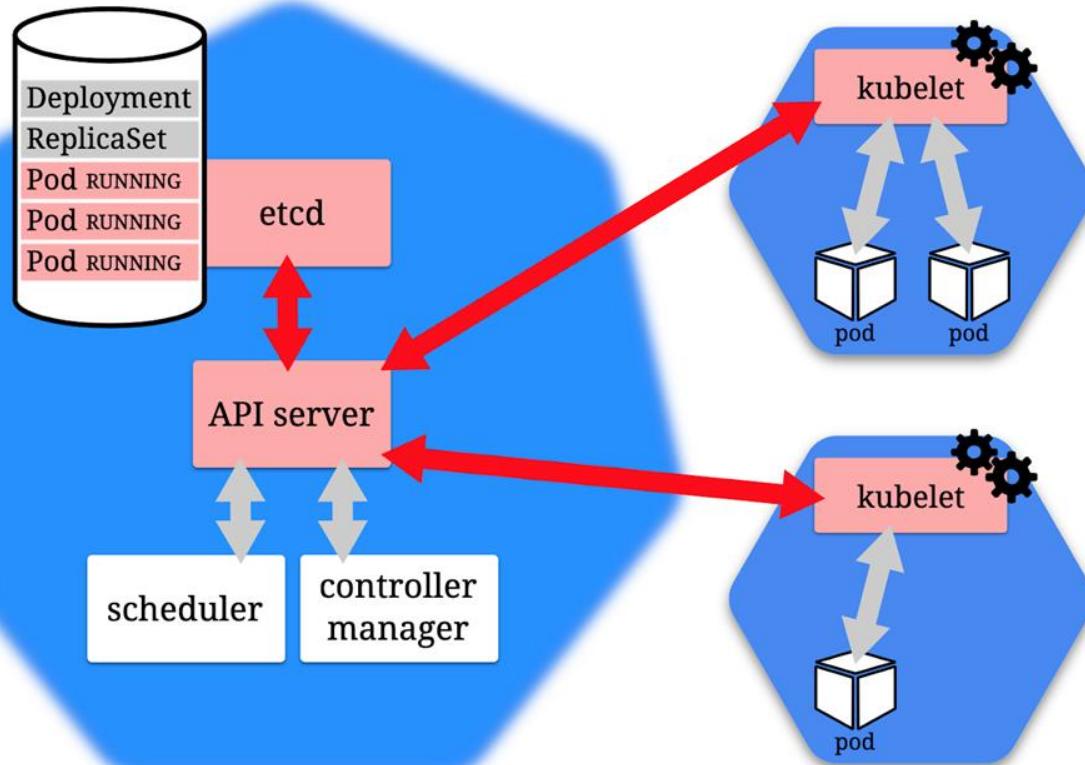
WORKER NODES



DEVOPS

CONTROL PLANE

WORKER NODES

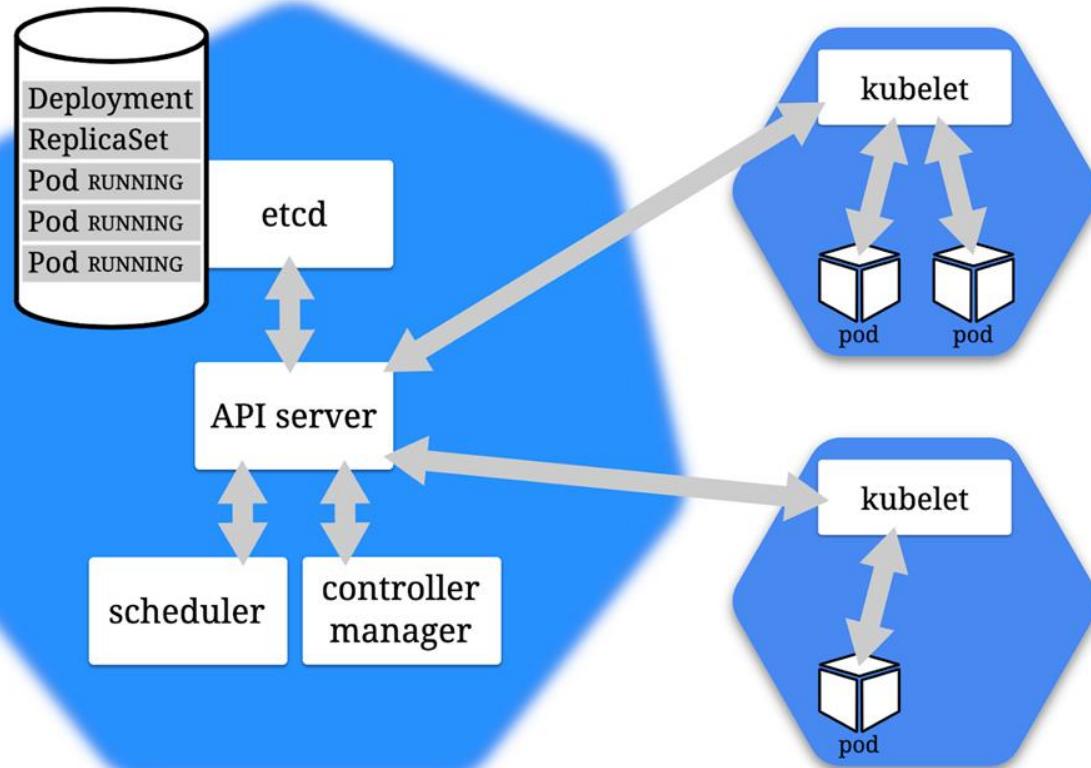




DEVOPS

CONTROL PLANE

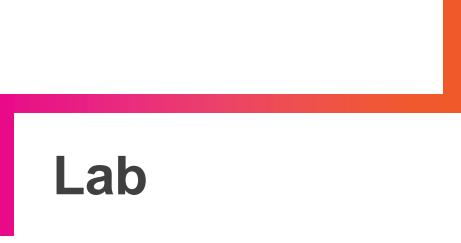
WORKER NODES



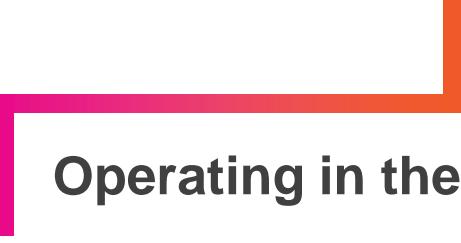
kubectl

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

Demo



Lab



Operating in the Cloud

Monitoring Across Hybrid Cloud

- Monitoring & logging are key considerations in any Cloud environment
- Systems (you hope) will be running around-the-clock – maximizing business benefit
- Unless you want to directly “babysit” those systems around-the-clock, you will need automated monitoring, logging and alerting to notify you of any issues
- Allows you to optimize handling for those exceptional cases when there is a problem

Monitoring Across Hybrid Cloud

- Capabilities exist to log and aggregate log data from the public Cloud
- There are likely already processes in place to monitor on-premise systems
- The goal is a strategy that allows you to pull that data together so you can analyze and make decisions holistically

Monitoring Across Hybrid Cloud

- Key tasks include:
 - Discovery – where are the critical data sources and how do I connect
 - Aggregation – bringing the data together in a systematic way
 - Normalization – converting data from disparate data sources into a canonical format
 - Security – data scrubbing (if required) and prevention of exposure of sensitive data
- Not just about identifying problems but also using the data to effectively identify opportunities

Monitoring Across Hybrid Cloud

Potential Challenges

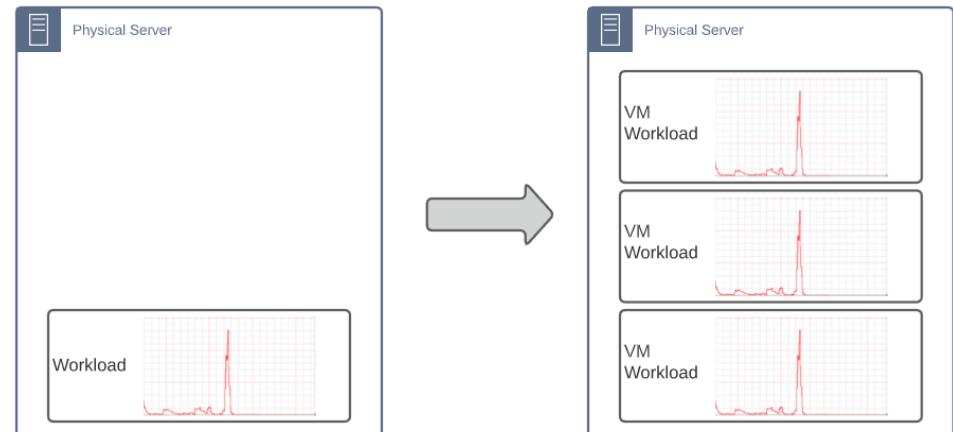
- You will need secure and performant connectivity between your on-premise and Cloud components
- Data formats may be very different between the different systems comprising your Hybrid Cloud environment
- You will need a strategy for gaining intelligence from the aggregated data while driving the benefit of that intelligence back into disparate systems

Virtualization & Orchestration

- Virtualization is not a Cloud-only concept, but the Cloud would not exist without it
- Virtualization enables an organization to get more value out of its infrastructure investments
- As we discussed, in the past, companies would try and estimate compute, network and storage capacity to cover 3 – 5-year growth

Virtualization & Orchestration

- Could result in two areas of challenge:
 - Underestimating – lack sufficient coverage to power the business
 - Overestimating – left with idle capacity, paid for but not adding value
- Virtualization enables the relatively quick spin up of right-sized infrastructure (and more of it in response to demand)



Virtualization & Orchestration

- Whether VM's, managed services or containers, more available instances require coordination
- Otherwise, the added complexity of "more" could impede rather than benefit
- Orchestration enables effective and efficient management as a *unit* so the "more" can be used to satisfy the business need

Virtualization & Orchestration

Potential Challenges

- Coordinating across multiple instances (sometimes very many) can be difficult – at either the infrastructure or application level
- Effectively combining the “many” into a pool of processing power, but still allow management at the individual instance level
- Optimal orchestration requires the ability to monitor the “many” and quickly respond

Elastic Scalability

- As highlighted previously, one of the main “draws” for Cloud is the ability to quickly scale up or scale down workloads
- In concert with virtualization & orchestration, the Cloud allows the automated spin up of “more” to handle:
 - Response to a specific schedule event (e.g., seasonal demand)
 - Response to an alert from a monitored event indicating that current configuration is being taxed with volume (using multiple metrics)
- It is elastic because the platform supports both scale up and down
- Key to optimizing cost vs. capability – paying for only what you need when you need it

Elastic Scalability

Potential Challenges

- Being able to determine what is needed and when can be challenging
- In the Hybrid Cloud environment, scalability may look different depending on whether you're talking on-premise services vs. Cloud services
- Determining optimal what & when may require usage data that you don't yet have with a newly deployed system
- Balancing capability against cost and ensuring "just enough"

Business Continuity/Disaster Recovery (BC/DR)

- A BC/DR strategy enables a company to plan for continued operations even in the face of a regional disaster
- Usually geographically-based – instances of services existing in *both* a primary region and in another physically-separated, secondary region
- That way, if the primary region goes down (for whatever reason), theoretically the company could continue to do business
- Doesn't have to be a permanent issue – could be a transient failure

Business Continuity/Disaster Recovery (BC/DR)

- Design and operational considerations:
 - Latency – because of physics, data can only travel over-the-wire at a certain speed
 - Active-Active or Active-Passive – does the system require / support actively servicing requests in both geographic locations at the same time?
 - Cost – depending on the configuration, a company may be required to pay for 2x the infrastructure

Business Continuity/Disaster Recovery (BC/DR)

- Most public Cloud platforms support “stickiness” to the region that is geographically closest to the request (to minimize latency)
- Two key concepts relative to data:
 - RTO – Recovery Time Objective (how much downtime can I absorb?)
 - RPO – Recovery Point Objective (how much data loss can I absorb?)

Business Continuity/Disaster Recovery (BC/DR)

Potential Challenges

- As discussed, latency can be a challenge – will a secondary region perform at the level needed to meet your SLA's?
- If the profile is Active-Active, it can be challenging to coordinate data collection and intelligence gathering across the two regions
- If the profile is Active-Passive, what is the process for spinning up the secondary region, how do you keep data in sync (and then undo once the disaster scenario resolves)?
- As with elastic scalability, balancing capability against cost and ensuring “just enough”

Endpoint Protection & Security

- The services exposed by a company used to provide its business value are critical
- The data consumed by a company in the provision of that business value could be very sensitive
- There are multiple regulations in place requiring the protection of sensitive data (e.g., PCI, SOX, HIPAA and GDPR)
- Failure to adhere to those regulations can cost a company significantly – either in actual \$'s or in reputation (which can be more damaging)

Endpoint Protection & Security

- The issue is not only one of data security – there are “bad actors” that work to take down sites and services
- One of the ways that service can be hindered is through a DDoS (Distributed Denial of Service) attack
- For DDoS, attackers will attempt to “flood” a service with so much bogus volume that it becomes unable to satisfy real business requests

Endpoint Protection & Security

- Most Cloud platforms provide services to help you protect against a DDoS attack
- Can include API management services (subscriptions, key-based access, throttling)
- Web Application Firewall (or WAF) is another service provided by Cloud platforms to monitor, filter and block (if required) incoming traffic

Endpoint Protection & Security

Potential Challenges

- The threat and regulatory landscapes are constantly evolving
- In a Hybrid Cloud scenario, creating a solution that gives you comprehensive protection across your on-premise resources, your Cloud resources and the connectivity between the two is not trivial
- Optimal application security and infrastructure security requires planning and specialized skillsets
- Good architectural practices (e.g., Least Privilege and Secure-by-Default) can help limit the “blast radius”

Assessing Security Risks

- To secure a solution, attack surfaces and potential threats must be identified
- Common practice utilizes something called threat modeling
- Includes modeling and analyzing possible attack vectors based on application

Assessing Security Risks

- Risk assessment should account for different “zones” of execution
- Security requirements for device in remote oil field different from secure data center
- And, ideally, threat modeling would be executed during design & dev phases

Threat Modeling



Securing Data in Motion

- Security required as data flows through the ether between producer and consumer
- If attacker able to intercept information flowing between the two:
 - Potentially exposes sensitive information contained within header or payload
 - Could allow insertion of alternate, damaging detail or control instruction

Securing Data in Motion

- Certificate/secrets-based Transport Layer Security (TLS) can be used to protect
- Highlights need to protect security keys
- Impact can range from trivial to devastating (depending on application)

Securing Data at Rest

- Aggregated data stored in plain text can create a vulnerability
- In previous topic on data management, goal is gained intelligence from the data
- If the data at rest has been compromised:
 - May lead to inaccurate conclusions from analysis
 - Could provide competitor or bad actor access to a company's competitive advantage
- As with "in motion", certificate-based encryption in storage is key

DevOps & Automation

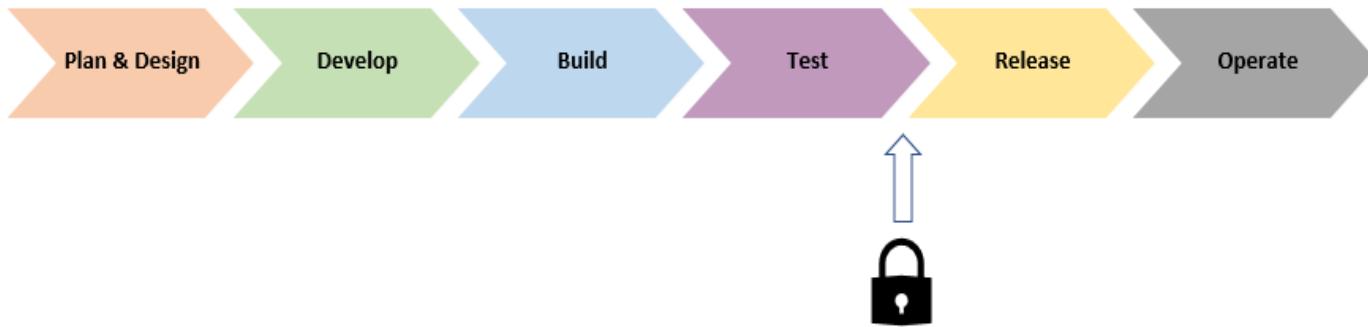
- As with other software applications, DevOps can provide lift
- Key principle of DevOps is automation
- Continuous Integration & Continuous Delivery can be applied as well

DevOps & Automation

- Presents opportunities to apply scripting to:
 - Automate onboarding, offboarding, and configuration
 - Deployment & configuration of Edge components
 - Deployment & configuration of Cloud services used to aggregate & analyze data
- Practicing principles of DevSecOps helps ensure security is “shifted left”

DevOps & Automation

What is often done:



What are the potential challenges with taking this approach?

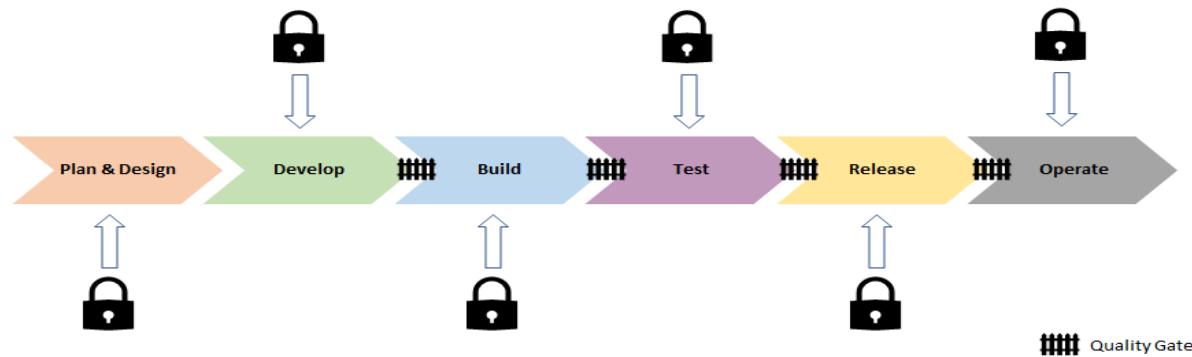
DevOps & Automation

Potential challenges:

- At this point, there may not be enough time in schedule to absorb change
- Activities required to remediate may be complex
- May require revisit of one or more previous phases to properly address

DevOps & Automation

Better approach:



DevOps & Automation

- Plan & Design – Threat modeling, data protection, and risk assessment
- Develop – SAST (Static Application Security Testing) tools
- Build – SAST and SCA (Software Composition Analysis) tooling
- Test – DAST (Dynamic Application Security Testing) tools, passive/active scans and “fuzzing”
- Release – Additional security-specific scanning, port scans, and log validation
- Operate – Monitoring & alerting, RCA (Root Cause Analysis)

DevOps & Automation

- Quality gates guard against moving security defects forward
- In true DevOps fashion:
 - Information gathered from early phases feeds into later phases
 - Lessons learned feed continuous improvement of overall process

Cost Management

- CAPEX vs. OPEX
- Azure → <https://azure.microsoft.com/en-us/pricing/calculator/>
- AWS → <https://calculator.aws/>
- Key is – ensuring accounting for ALL components
- Drives ROI (Return on Investment) & CBA (Cost-Benefit Analysis)

Thank you!

If you have additional questions,
please reach out to me at:
(email address)