RESTful Web Services

DevOps + Microservices + Cloud

Overview of DevOps:

Phases in DevOps:

Source code management (Git, Github, Bitbucket, GitLab)

Build management (Maven, Gradle, NuGet, webpack)

Repository management (Dedicated server for managing our binaries) (Jfrog Artifactory)

Test Automation (Junit, Cucumber)

Continuous Integration (Jenkins, TeamCity, Travis, Bitbucket, GitlabCI, CircleCI)

Continuous Delivery

Continuous Deployment

Infrastructure As a code (IAC)

Automate the routine stuff

Eg: Starting a server - in-person dependant.
Now, we are making use of automation tools to do things

With the explosion of Big data, infrastructure is managed through software code. Infrastructure code will be undergoing the test cycle.

Popular tools: Terraform, Ansible, Chef, Puppet

Continuous Monitoring: Splunk, Influx, Prometheus, Grafana

Helps in finding out the health statistics.

Why Cloud platform?

What are the major benefits of DevOps?
Cutdown the release time
Complex installation - Automation tools make easy

Business applications - not resilient - That was the problem. Build more resilient systems today Helps our application releases shorter. Time to market is a major benefit for the company and client.

Docker & Kubernetes:

The reason why to learn? Help to package its code and dependencies into one box and share it with the kernel.

Major problem:
Handing over the Dev team to the Operation team.

Bundle libraries, dependencies, and documentation and handle them over to the Operations team. Build process may not be successful. The advantage we get here is PODA (Package once and deploy anywhere). Here, transportation is not a problem at all.

Containerization: Package the application along with its dependencies. It runs in an isolated environment. It shares the space with other kernels and enjoys its isolation.

Kubernetes: Manage the orchestration between containerization. This is where Kubernetes plays a role. Helps to orchestrate between containers. That's why it is called Container Orchestration Platform. Discovering between containers. Managing the state, workload.

Resilient system - One system breaks down. We need to ensure that we will get the same system back. We need other sets of instances of the systems back.

Eg: A musician managing the music with his peers.

Jenkins: Pipeline. Everything has to be built. Here, developers commit their codes. Managing the infrastructure is done through code.

Role of Kafka: Message broker.

Overview of Microservices:

What exactly are services?
Business components/ functions which will be exposing the functionality.
Eg: 3 layer architecture: UI layer, and database.

Client application: Mobile application (Android system/ IoS system)
Browse-based application - Java, Angular, NuGet framework.

Services can be written through any language (Java, Python, C, or C++). It should be accessible by any other clients.

Service-oriented architecture:

How to ensure that our services can be accessible to all other clients?
RPC - Remote Procedure Call.

Limitation:
Same environment.

Common object request architecture:
Instead of using RPC, we are using IOP (Internet over Protocols). Accepted by large community members. Object management group (C++, Python, Java) - Communities who accepted IOP as a standard.

SOP: Simple object access protocol: Provides greater flexibility as its platform is independent. Advantage: Independent of technologies and systems. Standardized by W3C.

Traditional system: Pearl, C, or C++. We can expose the services by SOP. Share our services to clients independent of our stacks.

There is a difference between SOP-based and RESTful architecture. The difference between SOAP and REST is here.

The disadvantage of SOAP:
Format lengthy (XML format), define the envelope.

Pass the API key -> To access the business functionality.

Boomerang - SOAP & REST API (add this to the chrome)

SOAP request - Envelope, header, body - define payload (weather parameters).

What is RESTful architecture?
Multiple clients - handle the traffic. What do we need to do? Load balances
Hyper - media-based system
Client-server based

The focus is mainly on how to provide services to different clients. **Representational State Transfer Application Programming Interface** is more commonly known as REST API web service. It means when a RESTful API is called, the server will transfer a representation of the requested resource's state to the client system.

Stateless communication: Every request is complete enough to be understood by the client.
Uniform interface
Layered system

What is hypermedia-based?
Different forms are available XML or JSON.

Transfer of object state from one service to another. That is called resources. Resources are surrounded by verbs/ payload (user id, id, title, body). Representation has metadata: HTTP Headers, header name, header value. This tells me that I am sending JSON data. Interconnected by hyperlinks.

None of the instances will be overloaded and transfer the pieces of information.

Eg: E-commerce application.
Accounts, orders, managing settings, promotion, inventory, products.

How is REST designed?
Action-specific resources.
Noun-only resources.
Verb: GET, POSTS, PUT, DELETE.

For calling web services -> We need to provide a data input (base resource URL + data input)

Yet Another REST Client - Add to the chrome
You can try to use methods here.

Data input:
Path parameters: Provided by /
Query parameters: ?tilte= qui est esse
Eg: https://jsonplaceholder.typicode.com/posts?title=qui%20est%20esse
Resource representation
Headers

Resource Representation format:
JSON
XML

HATEOAS Principle:

Hypermedia as the Engine of Application State
How do I know in which parameter data input needs to be sent - This can be understood by this principle.

HTTP Methods:
GET - Read, possibly cached
POST - Update/insert/create without ID
PUT - Update/insert/create with ID
DELETE - Remove
HEAD - Read headers, has the version changed?
OPTIONS - “List the allowed” Options - (Used when querying), patch operation - partially update.

HTTP status codes
1xx - informational - Request received, continuing process
2 xx - Action successfully received, understood, and accepted
3xx - Redirected - The client must take additional action to complete the request
4xx - Client error - request container bad syntax or cannot be fulfilled
5xx - Server error - Server failed to fulfill a valid request or method not allowed.

Microservices:

Difference between microservices and services?
Microsized services

Monolithic service - large, one single server. With a large codebase and fixed stack, dependencies with technologies. Scalability is a big challenge.

Eg: The gifting season. More instances for the product will be more.
Eg: For a promotion service, it has dependencies with 3rd parties and it failed. This brought our whole system to come down. For any minor change, we need to perform the whole build. This is where we will be moving to the microservices-based system.

Reason to move to microservices: More agile, time to market, better maintenance, client demands on high performance with zero downtime, flexible.

Create a smaller lightweight system. All these are part of a single server. Logically grouping them and making them as an independent system. Independently managed, built, tested, deployed, changed, storage and shared across teams that are interested in. Fixing issues is much faster.

Accounts services is an independent service. It has its database. We can facilitate lightweight communication between these services. All these services can expose their functionalities as API to other services.

How to identify what goes into the microservices? For this, we need to understand the design

Why are we looking at Microservices now?

- Need to respond to change quickly
- Need for reliability applications (Time values here. If the application is down for a few mins, huge loss in revenue)
- Business domain-driven design (new principle - as come out in the new format - key for microservices - with applying this we will know how to use it)
- Automated test tools (manual test is rigorous and time-consuming & resources consuming)
- Release and deployment tools
- On-demand hosting technology
- Online cloud services
- Need to embrace new technologies
- Asynchronous communication technology (Kafka)
- Simpler server-side and client-side technology

Benefits of microservices:

- Shorter development times
- Reliable and faster deployment
- Enables frequent updates
- Decouple the changeable parts
- Security
- Increased uptime
- Fast issue resolution
- Highly scalable and better performance
- Better ownership & knowledge
- Right Technology
- No fixed technology stack
- Enables distributed teams

Microservices Design Principles:

- High Cohesion
- Autonomous
- Business Domain centric (Domain-driven design)
- Resilience
- Observable
- Automation

High Cohesion:

- Single focus: Design services with a single focus.
- Single responsibility - SOLID principle, only change for one reason
- Reason represents - A business function, a business domain
- Encapsulation principle - OOP principle
- Easily rewritable code
- Why - scalability, flexibility, reliability.

Autonomous:

- Loose coupling
- Honor contracts & interfaces (fixed contracts - so that they should not be broken)
- Stateless
- Independently changeable
- Independently deployable
- Backward compatible
- Concurrent development (versions)

Business Domain Centric:

Apply domain models and how to apply bounded context

- Service represents business function - Accounts Department, Postage calculator
- Scope of service
- Bounded context from DDD
- Identify coundaries\seams
- Shuffle code if required - Group related code into a service, aim for high cohesion
- Responsive to business change

Resilience:

- Embrace failure - Another service, specific connection, and third-party system
- Degrade functionality
- Default functionality - able to apply and disconnect it / circuit breakers
- Multiple instances - Register on startup, deregister on failure
- Types of failure - Expectations\Errors, delays, unavailability
- Network issues - Delay, unavailability
- Validate input - service to service, client to service

Observable:

- System Health - Status, Logs, Errors (Monitoring tools)
- Centralized monitoring
- Centralized logging
- Used in analytics purposes as well
- Resolves the issue much faster

Why?

- Distributed transaction

- Quick problem solving
- Quick deployment requires feedback
- Data used for capacity planning
- Data used for scaling
- What microservices-based used
- Monitor business data

Automation:
Automate the whole process.
Tools to reduce testing

- Manual regression testing (complex, rigorous, quite cumbersome)
- Time took on testing integration
- Environment setup for testing

Tools to provide quick feedback

- Continuous feedback
- Continuous integration

Tools to provide quick deployment

- Pipeline to deployment
- Deployment ready status
- Automated deployment

- Reliable deployment
- Continuous deployment

Why?

- Distributed system
- Multiple instances of services
- Manual integration testing tool time consuming
- Manual deployment time consuming and unreliable

What consists of microservices?
In monolithic, all the services are connected to single data storage.

How do we split the databases in microservices?
How to manage configuration between microservices?
How to document microservices?

Where does complexity come from?

- Make sense of requirements
- Build a (relational) data model
- Identify relevant tasks and data tables. Map with data tables. - Unmanageable
- Build a user interface

- Close to what users wanted but…

Persons to identify and solve the problem:
- Business domain experts
- A person who did researchers
- People good at UI
- End users

Why is data-driven design so Intriguing?
- Captured known elements of the design process
- Organized into a set of principles
- Made domain modeling the focus of development
- Different approaches to building business logic

DDD is still about business logic? Yes
- Identify what is expected.
- Crunch knowledge about the domain
- Recognize subdomains
- Design a rich domain model
- Code by model

The secret dream of any developer: An all-encompassing object model describing the entire domain
Give me enough time and enough specs and I will build the world for you.

The main focus of DDD has shifted
Discover the domain architecture more than organizing the business logic
Domain model remains a valid pattern to organize the business logic

Common summary of DDD:
Build an object model for the business domain
Consume the model in a layered architecture - 4 layers, business logic split and renamed.
Application layer and domain layers

What is DDD?
We design the system based on our knowledge of the domain. That's why we sit with our domain experts and find out the domain problems, then we try to apply them.

DDD has two distinct parts:
You always need one but can happily ignore the other
Valuable to everyone and every project
Analytical and strategic

Design-driven by the domain - key points

- Ubiquitous language - Definition and discovery
- Bounded context - Definition and discover
- Context Map - Design of top-level architecture

What is expected?

We have to release applications faster
Effortless implementation

Here, domain knowledge is needed!
This is knowledge about the domain which is operated by software. Understand what users are talking about and what they are expecting. We can enter into the problem space.

What if we lack domain knowledge?

The source is a specification.
Domain experts - provide insights about the domain. This gives out fruitful results.
Business analysts and domain experts - the art of getting the domain knowledge is the theme.

Ubiquitous language - what's that?

Able to understand the business domain

- The vocabulary of domain-specific terms: Nouns, verbs, adjectives, idiomatic expression, and even adverbs
- Shared by all parties involved in the project: Primary goal of avoiding misunderstandings
- Used in forms of spoken and written communication: Universal language of the business as done in the organization

Definition:

Natural language, not artificial
Comes out of interviews and brainstorming
Iteratively composed and refined along the way
Unambiguous and fluent: Meets expectation of domain experts, Meets expectation of technical people

Ubiquitous - Used everywhere

Words and verbs that truly reflect the semantics of the business domain

User stories & RFC
Meetings

Use the model as a backbone for knowledge

Start from user requirements
Eg: Voucher is the domain name

Synonyms like coupon or gift card are not allowed

At work definition:

- Set state of the game -> start/Pause the game
- Delete the booking -> Cancel the booking
- Submit the order -> Checkout
- Create an invoice -> Register the invoice

Bounded Context - What's that?

Ubiquitous language can change but it can't change as such. We need to have some boundaries. Beyond the boundaries which are changing, we remove that ambiguity and duplicates.

- Delimited space where an element has a well-defined meaning: Any elements of the ubiquitous language
- Beyond the boundaries of the context, the language changes - Each bounded context had its ubiquitous language

Motivation

What a microservice should be?

- Remove the ambiguity and duplication
- Simplify the design of software modules
- Integration of external components

How to structure our components?
Problem space (Identify the domain model) -> Solution space (identify the subdomain and bounded context)

Domain -> domain model
| |
V V
Subdomain -> Bounded context

Break the problem into the subdomain. Each of them is bounded by bounding context

Bounded context:

Ubiquitous language
Independent implementation (eg: CQRS)
External interface (to other contexts)

Each bounded context becomes one microservices. (It's a layered architecture)

Each bounded context will have a data access layer, application layer, presentation layer, storage layers, transportation layers, UI layers.

Eg: Book purchase application.

Bounded context
Starts off as a core domain concept
Internal models (Supporting)
Each BC has an explicit interface
Shared models for communication
Microservices = Bounded context
Ubiquitous Language
Belongs to a specific domain function. Also, used by all team members to connect all the activities of the team with software.

The core concept defines the language

- Ubiquitous language
- Natural core language

Used as a bounded context filter

- Concepts in context
- Concepts out of context

How to define the language

- Domain experts
- Software Developers

Bounded context as a technique
Key aspects\concepts of the domain
Concepts from bounded contexts
Core concept forms ubiquitous language
Rename supporting concepts\models
Move out of context concepts\models
Single concepts indicate integration

Bounded contexts become microservices

Driver
Deliveries
Customers

Aggregation (Decomposing)
When is it used? Data analytics
Adding functionality

- Combining services
- Used in addition to decomposing: bounded context method
- Reason for
- Reporting
- Enhanced functionality
- Usability for clients
- Performance
- When to decompose or aggregate?

Communication between microservices:
REST call is done
Independently deployable and changeable

Synchronous microservice communication
Event-based
Competing workers pattern
Fanout pattern

Asynchronous Microservices communication:
Async API call

Why asynchronous microservices?
Decouple the services. Why? - for better user experience
Event-based:
Transaction/action as an event
Messages using a message broker
Decouple client and service
Queuing pattern
Async API calls
Request/acknowledge using callbacks
Other options
Hangfire

Request/Response
Client calls service
Service carries out the task
Client receives response

Synchronous
Traditional for request/Response
Calls wait for a response(blocks)

Asynchronous
Possible for request/response
Registers a callback and client unblock
The response arrives on callback

Request/Acknowledge with Callback

Client -> API

1. Incoming request with callback info
2. Start as a background task
3. Return acknowledgment
4. Update status using callback

API vs work based

Functional requirements
Autonomous microservices principle

Loosely coupled
Independently changeable
Independently deployable
Support and honor contracts
Technology agnostic API
Stateless API

API architectural style
Pragmatic REST
HATEOS (True REST - every service will have a link to other services)
RPC
SOAP

API architectural pattern

- **Facase pattern**

A single interface to a subsystem
Each subsystem represented by a class
API request honored by the facade
Hide the complex system and focus attention on the main
Microservice API is just a facase

- **Proxy design pattern**

Placeholder for another object
Used to control access to another object

Another object could be an external API
The proxy object doesn't contain business logic
The proxy object is a wrapper that provides
Simplification
Transformation
Security
Validation

Hide the real object/services and expose the functionality as SOAP-based service. Apply certain validation and transform.

- **Stateless service pattern**

What are stateful services? Avoid for microservice
Stateless service pattern
No state information maintained
Clients maintain state
The state sent as part of the request

Advantages for microservices architecture
Scalability

Options

Traditional ACID transaction
Atomicity, consistency, isolation, and durability

Two-phase commit pattern

- ACID is mandatory
- CAP theorem: Choosing consistency

SAGA pattern

Two-Phase commit
Pattern for distributed transaction
The transaction manager manages the transaction

Prepare phase
Transaction manager asks to prepare
Voting phase
Transaction manager receives votes

Centralizing access to microservices
-API Gateway

Why?
Distributed functionality
Distributed data
Distributed security

Complexity for client application
Consolidating API data
Consolidating API functionalities
Managing security

Complicated customer experience

- API as a product

```
PS C:\Users\vsriniva> docker version
Client:
 Cloud integration: 1.0.17
 Version:           20.10.8
 API version:       1.41
 Go version:        go1.16.6
 Git commit:        3967b7d
 Built:             Fri Jul 30 19:58:50 2021
 OS/Arch:           windows/amd64
 Context:           default
 Experimental:      true

Server: Docker Engine - Community
 Engine:
  Version:          20.10.8
  API version:      1.41 (minimum version 1.12)
  Go version:       go1.16.6
  Git commit:       75249d8
  Built:            Fri Jul 30 19:52:31 2021
  OS/Arch:          linux/amd64
  Experimental:     false
 containerd:
  Version:          1.4.9
  GitCommit:        e25210fe30a0a703442421b0f60afac609f950a3
 runc:
  Version:          1.0.1
  GitCommit:        v1.0.1-0-g4144b63
 docker-init:
  Version:          0.19.0
  GitCommit:        de40ad0
```

```
PS C:\Users\vsriniva> kubectl version
Client Version: version.Info{Major:"1", Minor:"21", GitVersion:"v1.21.5",
GitCommit:"aea7bbadd2fc0cd689de94a54e5b7b758869d691", GitTreeState:"clean"
, BuildDate:"2021-09-15T21:10:45Z", GoVersion:"go1.16.8", Compiler:"gc", P
latform:"windows/amd64"}
Server Version: version.Info{Major:"1", Minor:"21", GitVersion:"v1.21.5",
GitCommit:"aea7bbadd2fc0cd689de94a54e5b7b758869d691", GitTreeState:"clean"
, BuildDate:"2021-09-15T21:04:16Z", GoVersion:"go1.16.8", Compiler:"gc", P
latform:"linux/amd64"}
PS C:\Users\vsriniva>
```

Classwork notes are available here.

DAY - 2

Infrastructure as a code - Manage the infrastructure through configuration files

Today's agenda:
Virtualization and containerization.

Command Query Response System (CQR):
Eg: Payment system
The same term means different for different systems.
Payment in Clubhouse is more about membership fees.
Payment in PayPal is more about the transaction.

Domain model: Guest, fees, members, and payment
Bounded context: Single focus and single responsibility
Context mapping: Technique that provides the view of the system. A web of bounded context. We have certain shared models. We query the data and get the response. Now, what do we mean is, the command is alter the state of the object but doesn't return the data.

Query the data: Doesn't alter the state but only return the data
Command: Alter the state but doesn't return the data

In traditional architecture, we have a presentation layer, application layer, domain layer, infrastructure layers and we have data access (called DTO - Data Transfer Objects).

Commands: Presentation -> Applications -> Domain -> Infrastructure
Query: Infrastructure -> Data Access -> Presentation

This is what is called a **Command Query Response** System. Here, we are building the separate part - command, and query.

In **Typical layered** architecture, we have a presentation layer, application layer, domain layer, and infrastructure layer. It's a **bi-directional layer.** This is what we follow.

Layered architecture is divided into two architecture:

1. We have the command model
2. Query model

When we apply the microservice system, we will apply these models.

Virtualization and dockerization:

The way we ship our software in a better way is a plan here.

What is Virtualization?
In a traditional system, how do we run our application?
We have a physical server, it will have an operating system(Windows/Linux).
Memory, CPU, network adapter - all these are shared between applications.

Can I share these things between applications? This is where the problem comes. This is where virtualization comes into the picture.

Virtualization:
In the computing world, we try to create a virtual computer, CPU, network resources, memory, storage, and hardware platforms.

Advantages:
In a single physical server, an operating system will be running including the memory, network, and storage adapters. Since all the resources running into server. How can we scale up if we need new resources?

Hypervisors: Provides abstraction on all your computing resources. I can have my app and OS. Not required to be bound to any server. We are able to run different apps and different OS. It uses the computing resources but does not actually know that it runs on a VM.

Virtualization is a hypervisor. What are hypervisors?
A virtual machine monitor(VMM) is computer software, firmware, or hardware that creates and runs virtual machines.

Many forms of virtualization
Server, storage, Network, Desktop, I/O, and application

What is a Virtual Machine?

What are guests and hosts? Those who run on top of the virtualization layer are called guests.

Understanding Virtual CPU, Memory, Storage, and Network:
Virtual CPU
Virtual Memory
Virtual Disk
Virtual Network

In the Host machine, we have computing resources that are CPU, we have 4 cores. This is available in the host machine.

I am seeing one of the VM that is being created. There are 2 VMs created? What does that mean?

It doesn't mean that these 2 VM will always be dedicated to those 2 CPUs.

Virtual Network: NIC - Network Interface Card

VM -> Virtual interface card -> Hypervisor -> Storage adapter -> Data store

Type 1 vs Type 2 Hypervisors

Type 1
Bare Metal Hypervisor. Here we have Hypervisor -> VM
Loaded directly on the hardware
Eg: Hyper-V, ESXi, KVM

Type 2
Hosted Hypervisor. Here we have OS -> Hypervisor-> VM
Eg:

How do you administer enterprise virtualization?
Through the virtualization management layers, we will do it.

Virtualization VS "The Cloud":

The IT challenge before virtualization?

Running the different applications on different machines, we were forced to buy many resources.

Advantage of virtualization:

- Lower the data center
- Lower the administrative cost
- Reducing energy consumption

Kernel-based virtualization - Handle the interaction between OS and hardware. The guest is running inside our individual kernel. The kernel should have the same configuration as a host.

Hypervisor virtualization - Directly runs on the OS.

Shared virtualization - Creates a ring and runs

Root based virtualization - Handles all the libraries

Containerization: Lighter version compared to virtualization since they share the kernel

We look for something better. What does that mean?

Even though we get some benefits from virtualization. The major things we talk about are effective ways to make use of resources.

With hypervisors, the amount of resources we share is effective. The advantage we get is mobility.

Why do we want to migrate to virtualization?

- Increased uptime
- Avoid downtime due to maintenance

Connecting to virtualization is the best choice? Yes

Why should we move to containerization?
As a DevOps best practice, today containers play a major role in software delivery and a strong pipeline through which we deliver the software fast.

What is happening is, in our day-to-day life, “Shipping containers”.

Do we really need a big ship to carry a truck of goods/load? Waste of time and resources

Containers are going to address this. The problem is while shipping the software to the operations team, the build is used to break down. Why?
Different versions
Resource complex

Handing over the code from the dev team to the QA team was complex and to the ops team was a big task.

Containers will have standard tools to ship the application. Give containers to them and they will use them in the production system. This can be done with help of docker.

Build, ship, and run it anywhere - Docker

Containers are processes (PODA - Package Once and Deploy Anywhere)

Advantage:

1. Containers are more secure - uses Linux security
2. Uses control group - to ensure and have some dedicated resources
3. Starving could be avoid

Containers are isolated processes like VM. Containers/images are immutable. It is able to scan for any immutables. Our software supply chain is more robust.

Containers are lighter & linear than VM. Why?
Containers don't require a guest operating system. Easy to design their infrastructure. Each docker system becomes a host.

Containers are processed with much more isolation. Everyone has access to everything. We need resources to handle it. Whatever we need, we make that as apps and run inside a container.

Since they share the kernel of operating systems,

We bundle the application together and use it in the form of containers. Everything will be shared with the underlined OS. In simple terms, I like to say that it is easy to install the application. Install the docker and share the resources.

Docker is an isolated process. Namespaces are like databases. Namespaces are like lists. Type of namespaces is like different types of lists. Images play a major role.

Where do we find the third-party module from in python? We will download from a common repository. Here, we have a docker hub where we have the common repository. From here we can download and use it. I don't need to run any OS. Just pull the image and make use of it.

Image: Container images
Image is nothing but a blueprint from which we can run it inside the containers.

Image vs containers layers:

Image (Nginx):
Nginx - load balancers

Layers created:
OS -> Framework (nginx) -> Application (solitaire)

Container
OS -> Framework (nginx) -> Application (solitaire) -> Container /Run (writable layer)

Build agent images

.NET: OS -> Application name -> NuGet (libraries)-> MS build (build management tools)

Java: OS -> Application -> JDK -> Maven

Single Machine:

Why lightweight? We don't have a guest

Hardware -> OS -> Containers

Can we effectively manage containers now? Yes

Set up in a cluster, so that resources not in use can be shared with others? Yes, to avoid unutilized resources won't get wasted.

Why forest of services? Here we think about using containers.

Containers run on different nodes.
How do we manage these containers which are running in different nodes? Health statistics? This is where we use Kubernetes - Containers orchestration platform. Here, we don't need to support multiple OS.

Traditionally software installation in production environments (DLL, shared libraries) - All these **formats become irrelevant.**

Docker architecture
A system where we have docker installed is called a docker host. It is aware of Linux OS. It is able to share the namespace (databases, control group, layering capability, ports, Device mapping, process ID) - we have other Linux OS functionality. This will be running into the containers.

Containers will not support the noisy neighbor syndrome with the help of control groups.
Advantage of control group:
It will allow the docker to limit the resources, CPU time, and amount of RAM to each container.

Container - D - component of docker. Higher-level functionality
Run C - lower-level functionality of the container.

Both are developed by the container network foundation. Container runtime is responsible for the run of docker.

Docker engine: Library for network, and connectively

Docker CLI - communicate to the docker engine by REST API cal. Make a REST call to the docker engine and perform the operations.

Go to the docker hub. Check here.

Command:

```
PS C:\Users\vsriniva> docker run centos ping -c 5 127.0.0.1
PING 127.0.0.1 (127.0.0.1) 56(84) bytes of data.
64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=0.311 ms
64 bytes from 127.0.0.1: icmp_seq=2 ttl=64 time=0.018 ms
64 bytes from 127.0.0.1: icmp_seq=3 ttl=64 time=0.024 ms
64 bytes from 127.0.0.1: icmp_seq=4 ttl=64 time=0.027 ms
64 bytes from 127.0.0.1: icmp_seq=5 ttl=64 time=0.022 ms

--- 127.0.0.1 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4179ms
rtt min/avg/max/mdev = 0.018/0.080/0.311/0.115 ms
```

```
PS C:\Users\vsriniva> docker run alpine echo "My first statement"
Unable to find image 'alpine:latest' locally
latest: Pulling from library/alpine
a0d0a0d46f8b: Pull complete
Digest: sha256:e1c082e3d3c45cccac829840a25941e679c25d438cc8412c2fa221cf1a824e6a
Status: Downloaded newer image for alpine:latest
My first statement
```

```
PS C:\Users\vsriniva> docker run -d --name jsonplaceholder alpine sh -c "while :; do wget -qO- https://jsonpla
ceholder.typicode.com/posts; printf '\n'; sleep 5; done "
ea55782a985659afd8bcd71693f0ac3f94a82af7f9bdfa150975173667f653cb
PS C:\Users\vsriniva> docker ps
CONTAINER ID   IMAGE                  COMMAND                  CREATED          STATUS          PORTS     NA
MES
ea55782a9856   alpine                 "sh -c 'while :; do …"   20 seconds ago   Up 21 seconds             js
onplaceholder
```

```
PS C:\Users\vsriniva> docker run -d --name nginx -p 9090:80 nginx:alpine
Unable to find image 'nginx:alpine' locally
alpine: Pulling from library/nginx
a0d0a0d46f8b: Already exists
4dd4efe90939: Pull complete
c1368e94e1ec: Pull complete
3e72c40d0ff4: Pull complete
969825a5ca61: Pull complete
61074acc7dd2: Pull complete
Digest: sha256:686aac2769fd6e7bab67663fd38750c135b72d993d0bb0a942ab02ef647fc9c3
Status: Downloaded newer image for nginx:alpine
c161468b6075f4171a0641cd6d5d55611631159fe747be4a1757d6f493e2c7b8
```

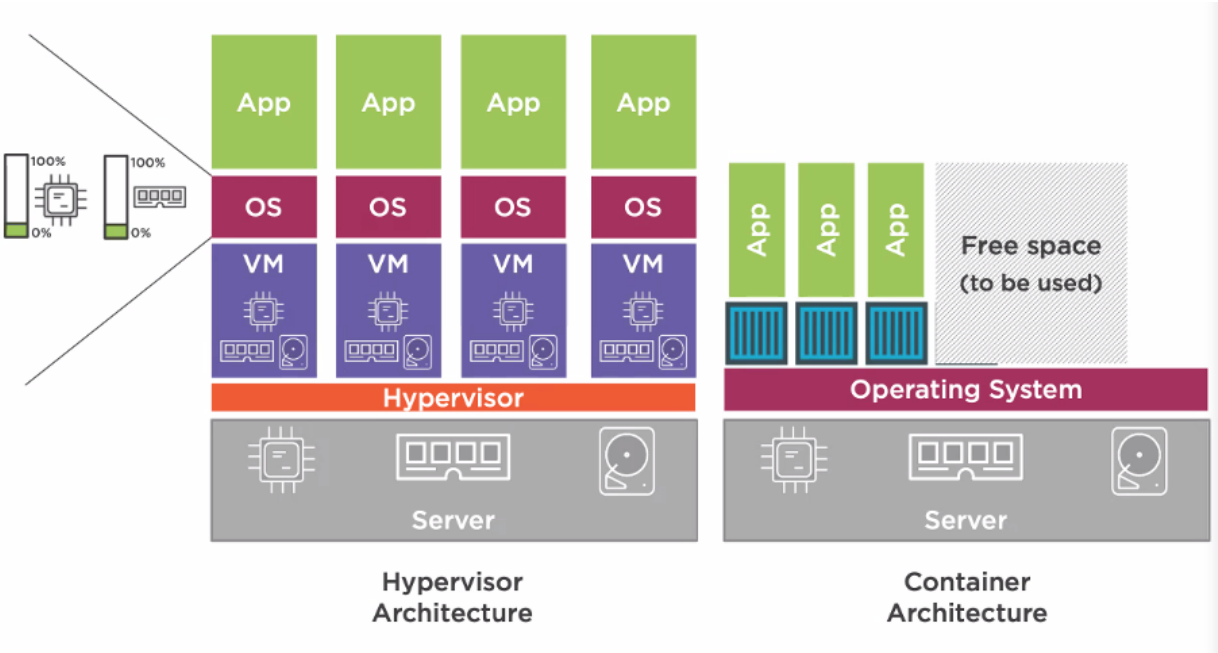
```
PS C:\Users\vsriniva> docker exec -it nginx sh
/ # curl -4 localhost:80
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
html { color-scheme: light dark; }
body { width: 35em; margin: 0 auto;
font-family: Tahoma, Verdana, Arial, sans-serif; }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
/ #
```

Namespaces is an abstraction consisting of the filesystem, network system, PID, or system group ID. Additional namespaces could be created or added to the existing containers.

Every mount can be either attached or added. Assuming that you have a host file system, what will it do now?
It will be copied to another file system. Remember the docker D gets initialized, similarly, you try to run one more subprocess, it gets created.

In Linux, **control group** - to isolate the resources usage of containers. (CPU, network, memory). Using a control group, we can avoid the parent 1 consuming all the CPU time or reserving a massive amount of RAM.
Run-C - lightweight and helps to get the access.

On the VMs only we will install the docker containers.



Whole operating system is now available as a filesystem.

Image: Blueprint
Virtual process. Images are created and executed inside the container. They consume data and produce the data.

Volume will continue to exist even after the life time of the containers
Image contains a layered file system. What does it mean?
Monolithic block. Image can be considered as a stack of layers. They will have the changes with respect to the filesystem. This is used to create a virtual filesystem out of it.

They are immutable. What does it mean?
Once it is generated, the layer can't be changed at all. Only the deletion operation can be performed.

Custom application : solitaire
Base layer -> nginx -> solitaire.

What does the Base layer contain?
We have OS, Alpine, nginx, and on top of that we will add our custom application

One writable layer will be there. That is called a Container/Run. That will have read/ write permission. The writable container layer can be shared with other images.

Different ways to create a image:

1. By interacting with the containers (Addition/updating). Those can be converted into a image
2. Docker file
3. Just by importing into the top of the filesystem

How to create images effectively and make certain changes?

Tagging a image:
Provide a unique name and then push into the docker hub where it is publicly available. Used to version our images. It has a border meaning.
By default, docker pulls latest image

We can also push our images into the registry as well. You have to have a docker hub account.

How do we persist the data? Completely restricted on the data?
Containers are preferred to be stateless. Data will be persistent with the containers.
Docker volumes are able to persist the data beyond the lifetime of the container.

Potentially have a concurrency issue? Yes.
Issues like automocity will start. Same space is able to read and write and shared between all containers.

To avoid the inconsistency, we can share it in read only mode.
We usually work in a distributed environment.

How to build components which are loosely coupled?

We have multiple apps in multiple containers and we need to see how they communicate with each other?
We need a common registry to look up the DNS. To look up the load balance.

Rolling updates? Application updates or base image update. One of the means of updating is to use rolling updates. One of the functionality of the application is update. Since there are multiple instances running, how to ensure. If one service ends, it will be updated to a new instance and monitored for some time and then it comes down and takes care. In kubernetes we have this update

How to ensure that we need to send this and it reaches in timely manners?

Networking in containers: How will containers communicate with each other? What are the restrictions available?

Container networking model: Software implemented has to follow a sandbox. It helps to isolate containers from the outside world. It won't make sense.
Endpoint - controlled gateway. Helps to protect the containers. It is connected to a network container but not the actual container.

Here, we talk about the container to container communication.

docker network ls command:

Bridge network: Allows single network
Overlay: Docker swarm and Kubernetes.
Null -
host network - port is directly bonded to the host machine
Contain network - provided by CISCO

DAY 3

Docker Compose Manages Your Application Lifecycle

We need a technique wherein containers will communicate with each other. For that, we have a network. We have discussed this earlier. This is a single-path communication.

Kafka overview:

Docker-compose and Docker swarm:

Multi container setup - manage multiple containers where we will be able to run and orchestrate the container on a single host. In a continuous integration process, developers will be continuously committing their codes, and the QA team does its continuous process. To manage this, docker-compose plays a role.

Docker swarm:

A tool provided by the docker community which is meant for orchestrating containers. Different nodes and clusters we can set up.

An alternative to these is Kubernetes.

Ex of docker-compose - advantage - helps to manage containers together. The entire lifetime of the container is managed by docker-compose.

YAML - standard language for docker-compose - Ain't Markup language Or Yet another Markup Language

The advantage is that it has a key-value pair. We have a map/list in YAML. This file is given as an input to the container to manage it.

Docker-compose feature:

Manage the whole application lifecycle:

- Start, stop and rebuild services
- View the status of running services
- Stream the log output of running services
- Run a one-off command on a service

Multi services app - What does it mean?
where we can start and stop multiple services.

Need for docker-compose:

Nginx - web server - load balancer -> send to different node JS instances - configure and scale them -> redis (kind of a no sql database) and mongoDB

All these can be placed in a single docker-compose.yml file and managed. Limited to a single host network.

Workflow of docker-compose:

Build services -> Startup services -> Teardown services

Role of the docker compose file:

Docker-compose.yml(service configuration) -> docker compose build -> Docker images(services)

Docker-compose and services

Version: "3. x"
Services: node JS, mongo DS

Docker-compose.yml

We can also define volumes and networks in the same YML file. This is the advantage we get in the YML file.

Key Service Configuration Options:

- Build
- Environment - if we need to define proxy
- Image - download from docker hub
- Networks
- Ports - port mapping
- Volumes

Docker-compose.yml example:

```
Version: “3.x”
Services:
        node:
                Build:
                        Context: .
                        Dockerfile: node.dockerfile
                Networks:
                        - nodeapp-network
        mongodb:
                Image: mongo
                Networks:
                        - nodeapp-network

networks:
nodeapp-network
Driver: bridge
```

If we provide depends_on: then there is no need to mention the IP address

Restart: always Whenever the container is stale, it clears the odd data and creates a new instance.

DNS is configured by looking into the lookup address rather than the IP address.

Docker-compose:

Restart flag - restart the container, pull the containers. It will stop and restart the containers. If we make any changes to the yml file. Then configuration files change. After restart the changes will be affected/reflected. Like environment changes will be reflected.

Apache Kafka

Asynchronous-based - event-based.

Streaming platform

Why is it popular? Horizontal scaling without compromising efficiency. It will be written in Kafka. Others are written in Java. We need a Kafka server.

Highly scalable
The larger community uses it
The amount of penetration is nearly 70% and more compared to other competitors

What is Apache Kafka? More like a service bus. Helps to connect to a heterogeneous application. Message router/broker. It means that it is a message broker, routing the messages among servers fastly.

High throughput
Publish subscribe model
Have logs for commits - analyzing

Microsoft SQL Server, MongoDB, and MySQL -> Elastic search, Oracle and Hadoop.

What does a typical enterprise look like?

Database Replication and log shipping

- RDBMS to RDBMS only
- Database-specific
- Tight coupling
- Performance challenge(logs
- Cumbersome(subscription)

Isolates the producers and consumers - it follows the smart client model and broker model

ETL - extract, transform and load

Lots of custom development
Scalability challenges

Messaging:
Limited scalability
Smaller messages

Problems faced:

The high volume of messages? Is no throttle done? Slow consumption? No consumption?

To solve these problems, message broker Kafka comes into the picture.

Middleware challenges

Multi-write pattern
Message broker pattern

Single node and single broker
Single node and multiple brokers
Multiple node and multiple brokers

This guarantees the delivery of the message. In Kafka, there are three means to deliver the message.

1. Zero loss of message - object, map, or text message
2. Message can be sent again/ redelivered
3. Never redelivered. Once delivered they are not sent again.

Cluster: Kafka brokers - manage a set of brokers.

Zookeepers - It is no longer required. Cluster coordinator kind of registry in a cluster environment.

Destination of the message

We have a queue, one broker can run multiple topics. For each message, there will be an identifier - offset. There will be a partition and it is structured and it is done sequentially.

There is also a retention period. Messages should be consumed within this period or else it will be erased.

Next-generation messaging goals

- High throughput
- Horizontally scalable
- Reliable and durable
- Loosely coupled produces and consumers
- Flexible publish-subscribe semantics

Kafka confluent: Enterprise version:

https://www.confluent.io/?utm_medium=sem&utm_source=google&utm_campaign=ch.sem_br.brand_tp.prs_tgt.confluent-brand_mt.mbm_rgn.emea_lng.eng_dv.all_con.confluent-kafka-general&utm_term=%2Bconfluent%20%2Bkafka&creative=&device=c&placement=&gclid=CjwKCAjwzt6LBhBeEiwAbPGOgeYGzTdHq0LrE-ywbn3OF1N9UeO_V5_-3P9MpPp-4ldNHy4_PFeHXhoCxrwQAvD_BwE

Scalability is a big challenge in IBM MQ.

Vertical scalability: Add resources(CPU) to the same node

Horizontal scalability: Add the computing machines to the resources pool.

Queue: Partition. A broker can have multiple topics. An ordered sequence of messages was added to the broker.

Kafka found it by linkedIn and then moved to Apache software.

Why do we use multiple brokers? To have high data volume. Parallelism and redundancy,

Replication factor - this will help to identify the no of brokers in a replica.

Distributed System: Controller Election

Distributed Systems: The Cluster, getting work done - Reliable

Named feed or category of message
- Producers produce a topic
- consumer consume from a topic

Logical entity
Physically represented as a log

Replica - list of broker - list of topics will be replicated across multiple brokers
The partition will help us to achieve parallelism.

Event sourcing:
An architectural style or approach to maintaining an application's state by capturing all changes as a sequence of time-ordered, immutable events

Message content:

Each message has a:
- Timestamp
- Referenceable identifier

- Payload(binary)

Offset - maintained at a consumer side.

If there is no consumer for a message - all publish message - default retention period is 168 hours (7 days). It will be erased.

Each topic will have one or more partitions. It is the base for scalable. Because of this, we achieve high throughput.

How do we get our Kafka?
With a single cluster/node we have seen how to publish

https://kafka.apache.org/downloads - download

Kafkacon - Kafka connect image will be able to create a topics

Docker orchestration and Kubernetes:

What is orchestrate?
When we containerize the application, microservice-based architecture. For an external client, there are certain APIs exposed. The clients have to communicate by APIs.

Difference between the Docker swarm and the docker-compose?

Docker-compose - single host machine
Docker swarm - orchestre. Multiple containers are distributed across a network. They require communication.

Eg: if there is no conductor or director, then the beats are not organized correctly.

Instead of musicians, we have containers that have different apps running. Instead of music being played, we have containers running to communicate with each other.

A declarative way of service: yml file.

Run an application on a service. Orchestrators scheduled these containers in a certain way. Replication and global service.

Replication services - specific the service which is needed to run all the time.
Service discovery - API looked up

Take care of routing as well - enable load balancers so that not one service is overloaded with the requests.

In docker-compose, we add an option to scale, especially when we are running an application which is containers, - increase in the flow of traffic - if there is an increased workload - we will increase the instances. Configure communication in newly configured instances.

We need a dynamic in nature - scale up and down easily in a more meaningful way.

Resilience - is applied to containers as well - self healing - Recover the containers in the previous state.
Zero downtime is achieved here. Only a few instances will dig down.

Popular Orchestrator:

Docker swarm - docker community

Collection of nodes. The node can be a physical computer or a virtual machine.

Eg: Linux OS -> Hypervisor (which give me the virtualization) -> different VMs -> on these VMs we have Linux installed. (These are considered as hosts)

Similarly, we have on other similar VMs ….

Can we set up docker on these VMs? Does it make sense?

Why we are moving from docker-compose to docker swarm?
Orchestration platform. Collection of nodes - a swarm.

What does the docker swarm do? Manage all containers. WE need to have one manager node for high availability. We can have multiple followers/workers. They have a gossip protocol where they can communicate with each other. We wrap them and manage them with different containers.

What are Services? - describe the state of the application - Deployed to our swarm - manifest file describing what will be in the yml file. (version, services - port mapping, environment, networks, replicas)

Docker-compose is a single docker host. (driver, bridge, host, null)
Docker swarm - Containers running on different nodes and able to communicate with each other.

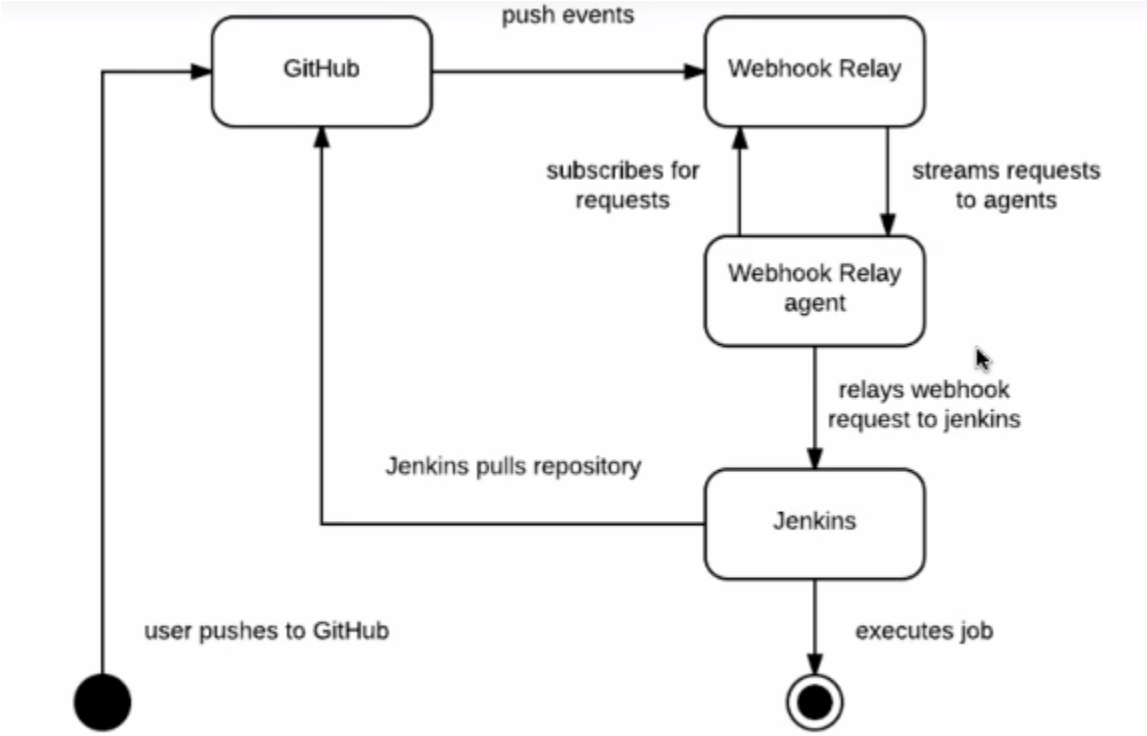
How do they communicate with each other?
Overlay network - enable communication that is running on different hosts/nodes.

We can elect our leaders. It has got synchronous communication to identify the communication like load balance.

```
Windows PowerShell
PS C:\Users\vsriniva\docker_training> docker swarm init
Swarm initialized: current node (m1utad9hd9e9ojes4yqkv3a1o) is now a manager.

To add a worker to this swarm, run the following command:

    docker swarm join --token SWMTKN-1-2yvjyjowrc7hvl6uscxsyskq0sxfgivrr66824554pzjddo811-aen2um1ubg5zisdzx7xoh2m9i 192.
168.65.3:2377

To add a manager to this swarm, run 'docker swarm join-token manager' and follow the instructions.

PS C:\Users\vsriniva\docker_training>
```

Whatever Kubernetes does we will get it in the docker swarm.

We have a webhook relay to connect through GitHub and Jenkins
https://webhookrelay.com/
Install webhook relay



Generate a public IP address, we will be able to router to other networks/systems.

https://labs.play-with-docker.com/

replicas: 6
update_config:
parallelism: 2

Replicas - 6 instances

Update config - Rolling update(what it is? - we will make some updates, patch updates - you want them to be added) will be updated in the batches of 2. After the 10s dealy, the other two instances will be updated. This will provide you with zero downtime.

It has to be the overlay network.

One stack can have n no of services.

Stack and services will be stored in the docker swarm manager. Then it will be shared with the docker workers.

Zero downtime deployement

Kubernetes - Coming from Google

Apache Mesos -

Marathon

AWS ECS

Microsoft ACS

Docker swarm - only works with docker community

Kubernetes - It works with any container platforms.
Go language
Manage, scale and all the things can be done
Container orchestrations platform
lightweight

We have the

kubernetes master - Kuberenetes master connects to the etcd through HTTP/HTTPS

Kubernetes Nodes - connects to master through http,

Kube control manager

docker image prune - Core engine
manage all schedules, data storages, expose APIs

Kubernetes Network

Key Features

- Service Discovery/ Load Balancing
- Storage Orchestration
- Automate Rollouts/Rollbacks
- Manage workloads
- Self-healing
- Secret and configuration Management
- Horizontal scaling
- More.

We are not using the name “slave” -> instead call it as followers

Kubernetes - the big picture

We have POD in each node.

What is POD and container? The POD is the one which will encompass the containers.

These POD will helps us to manage containers. It can have one or many. It can be distributed in many/ different nodes.

Running kubectl

Minikube

Advantage of service:

- PODS live and die
- Services abstract pod ip address from consumers
- Load balances between pods

Converting docker compose to kubernetes

kubectl version

kubectl get [deployments | services | pods]

kubectl run nginx-server --image=nginx:alpine

kubectl apply –f [fileName | folderName]

kubectl port-forward [name-of-pod] 8080:80

9663398670 - pradeep