

CLOUD COMPUTING

I



Introduction to Cloud Computing

- The US National Institute of Standards and Technology (NIST) defines the cloud as –
 - Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction.

Cloud

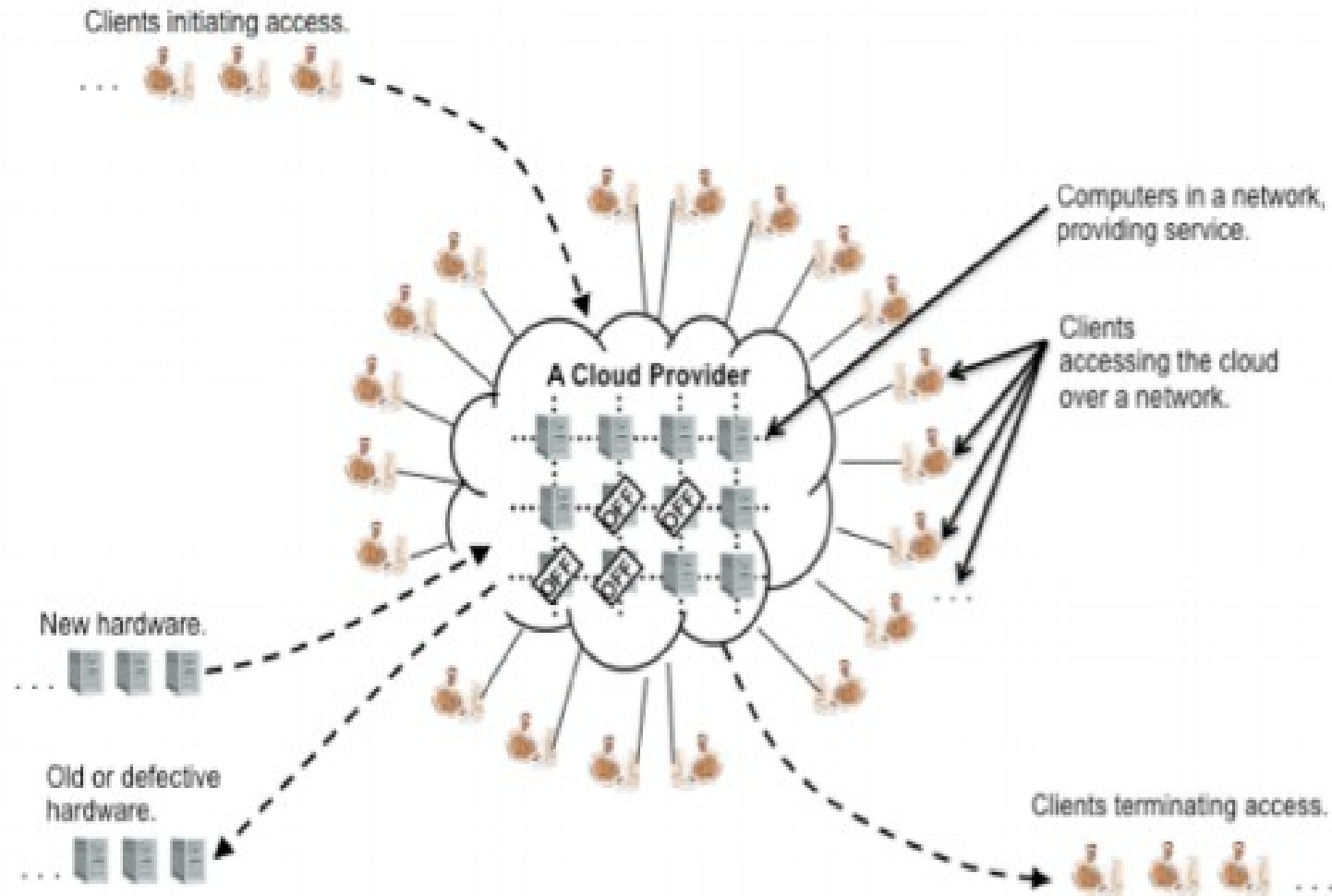


Figure 1: General Cloud and Subscriber View

Introduction to Cloud Computing

- cloud computing is simply a means of delivering IT resources as services.
- Almost all IT resources can be delivered as a cloud service: applications, compute power, storage capacity, networking, programming tools, even communications services and collaboration tools.
- Cloud computing began as large-scale Internet service providers such as Google, Amazon, and others built out their infrastructure.

Introduction to Cloud Computing

- It allows applications and services to run on a distributed network using virtualized resources.
- Also allows these to be accessed by common internet protocols and networking standards.
- End-user is not required to know the physical location and configuration of the system that delivers the services.

Introduction to Cloud Computing

- It delivers higher efficiency, massive scalability, and faster, easier software development.
- It's about new programming models, new IT infrastructure, and the enabling of new business models.
- more efficient large-scale cloud deployments that can provide the infrastructure for next-generation business opportunities: social networks, algorithmic trading, continuous risk analysis, and so on.

Boundary of a Cloud

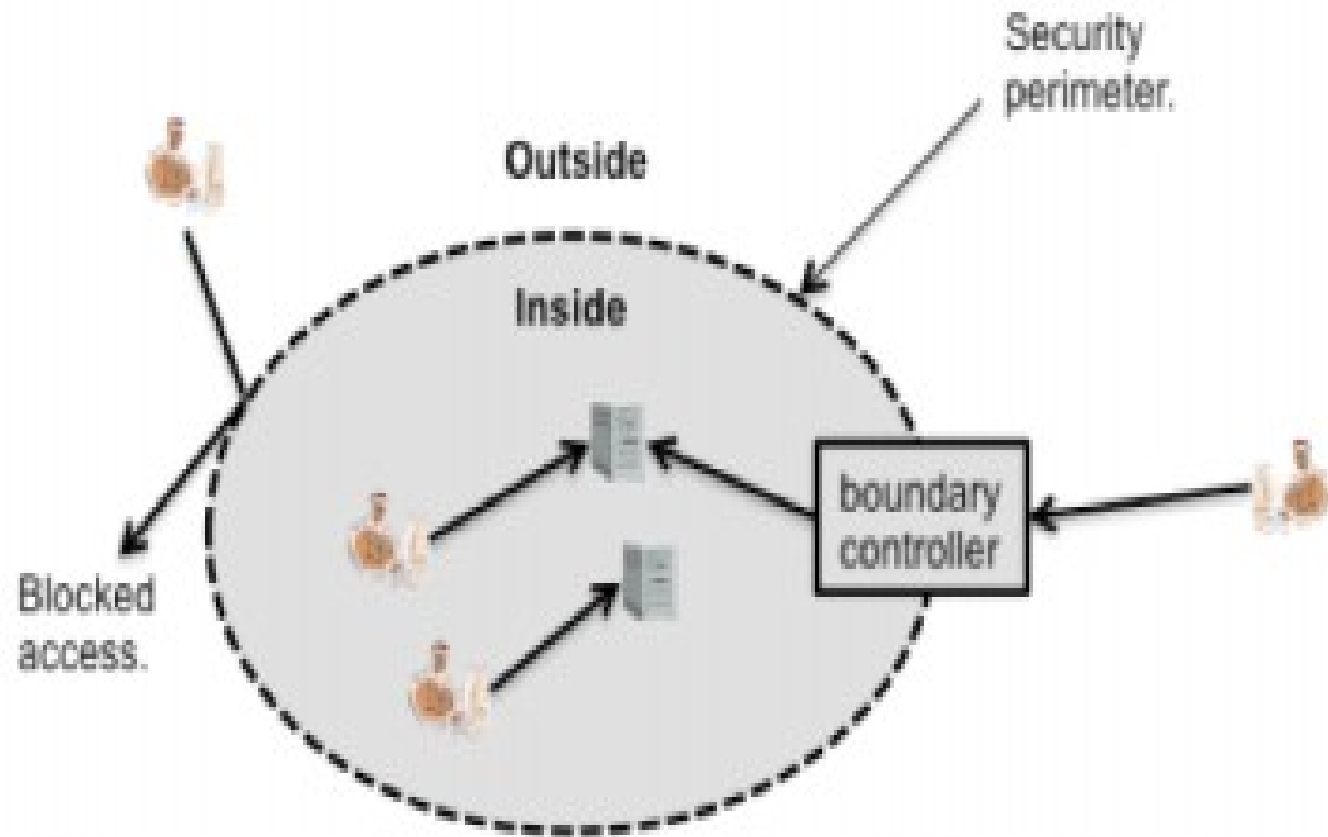
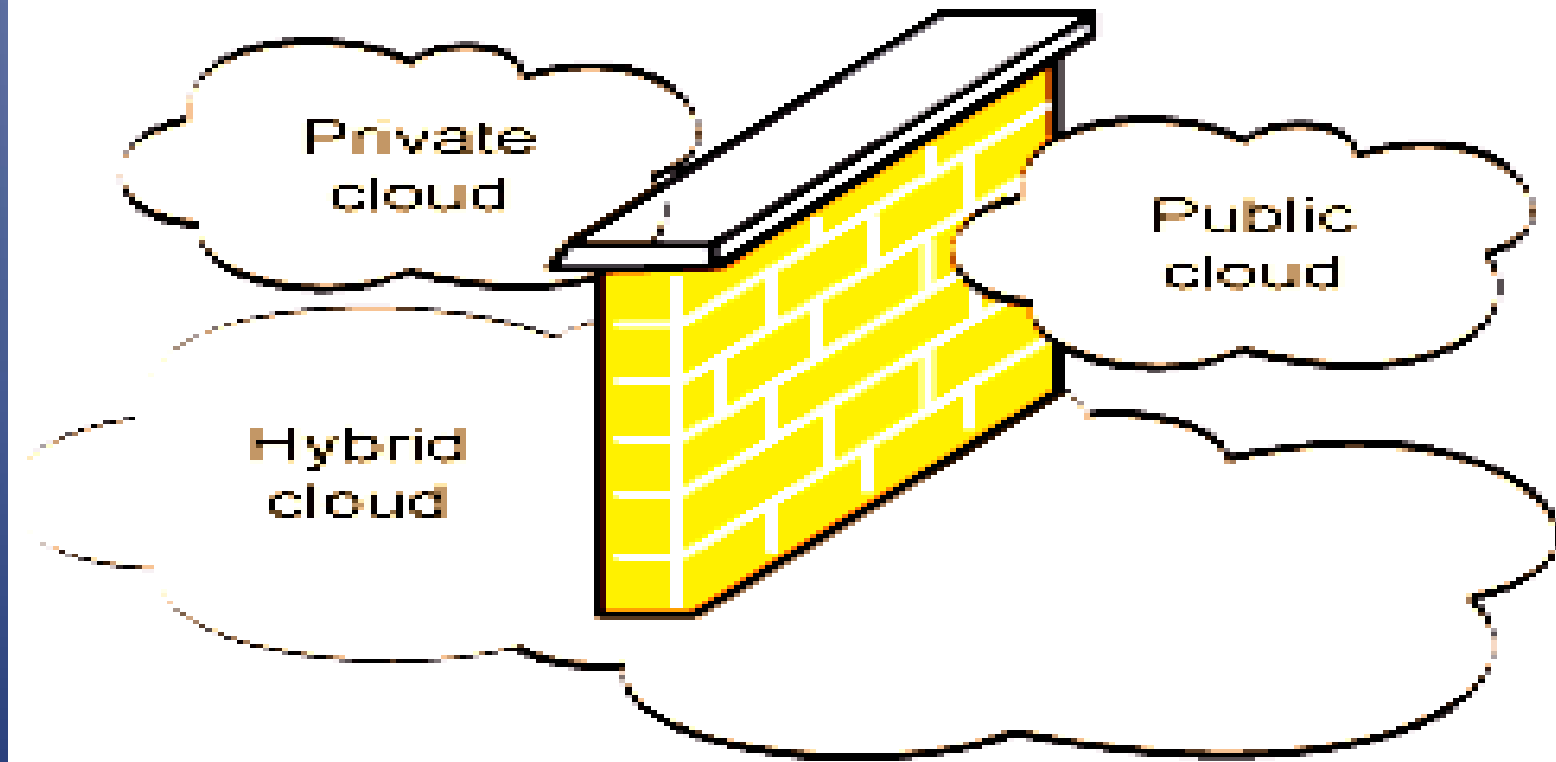


Figure 2: The Security Perimeter

CLOUD COMPUTING BASICS

Enterprise firewall



CLOUD Types

Cloud Types

- NIST has a set of working definitions that separate cloud computing into two types of models.
- Deployment model
 - Based on the location of the cloud and its purpose
- Service model
 - Based on the type of service provided by the cloud.

Cloud Types

- Deployment models.
- Private cloud.
 - an organization's own cloud.
 - It may be managed by the organization or a third party
 - may exist on premise or off premise.
- Private clouds are a good option for companies dealing with data protection and service-level issues.
- Private clouds are on-demand infrastructure owned by a single customer who controls which applications run, and where.
- They own the server, network, and disk and can decide which users are allowed to use the infrastructure.

Private Cloud

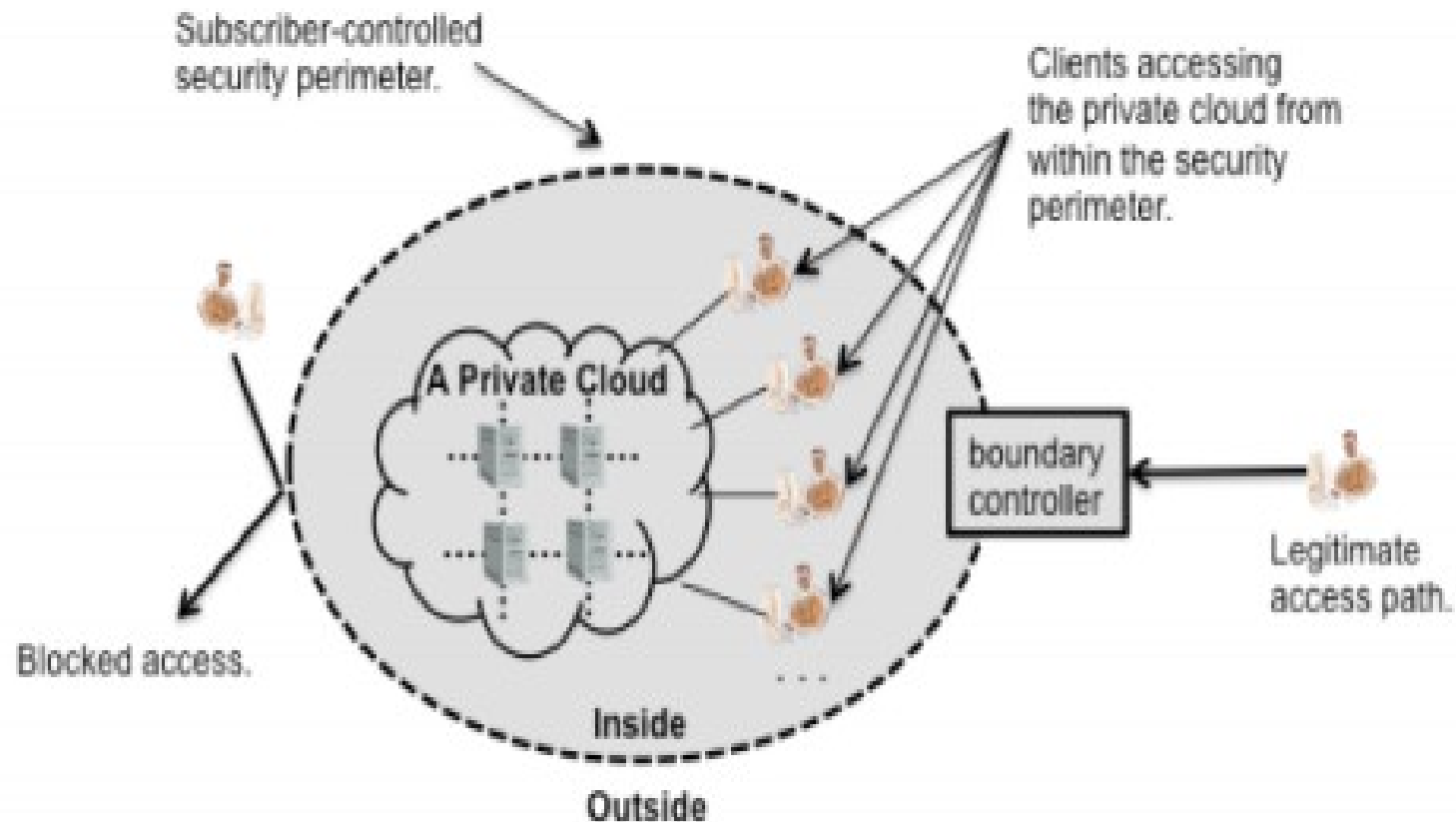


Figure 3: On-site Private Cloud

Outsourced Private Cloud

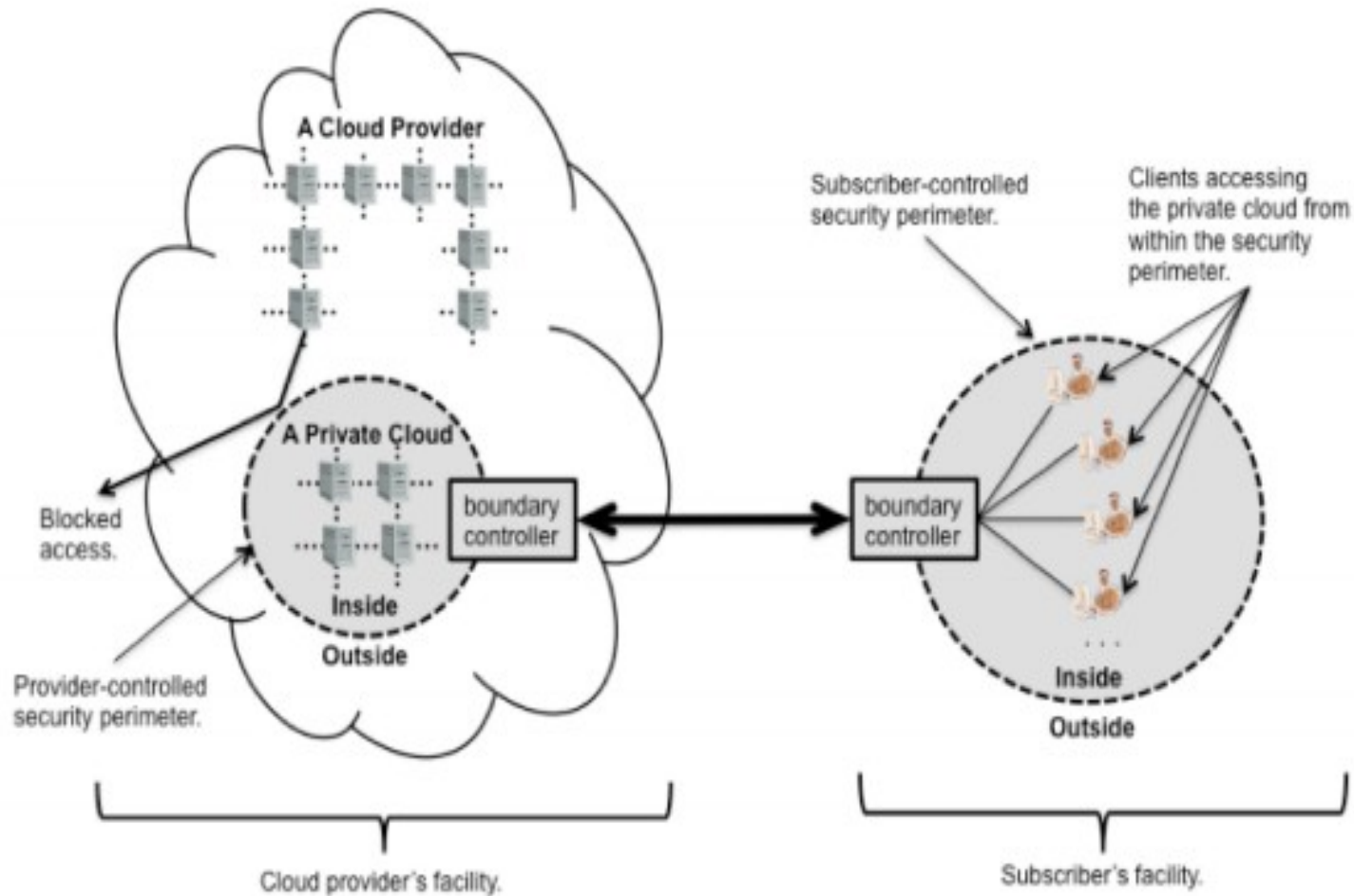


Figure 4: Outsourced Private Cloud

Cloud Types

- Deployment models.
- Community cloud
 - The cloud infrastructure is shared by several organizations and supports a specific community.
 - It may be managed by the organizations or a third party
 - may exist on premise or off premise.

On-Site Community Cloud

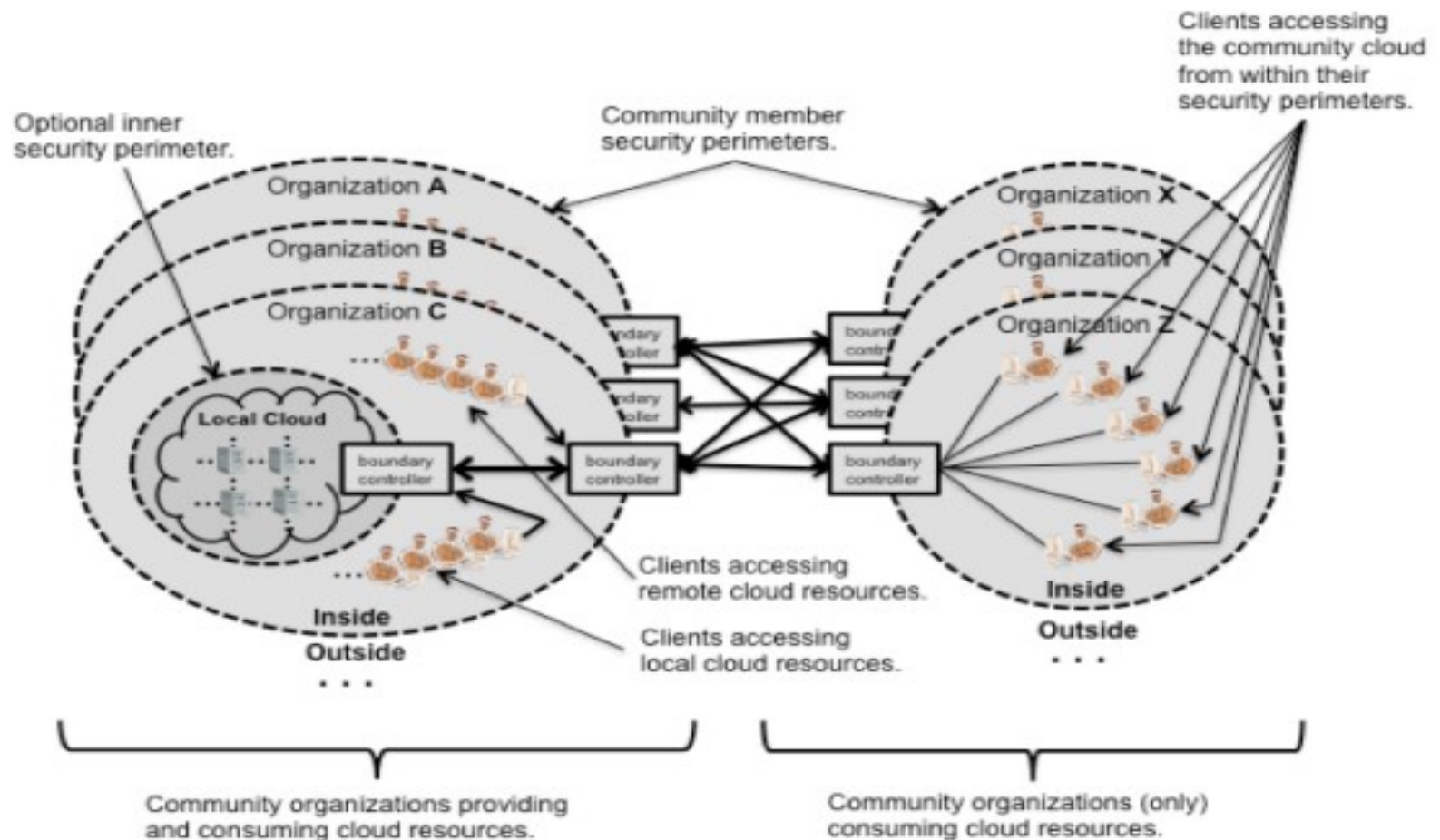


Figure 5: On-site Community Cloud

Out Sourced Community Cloud

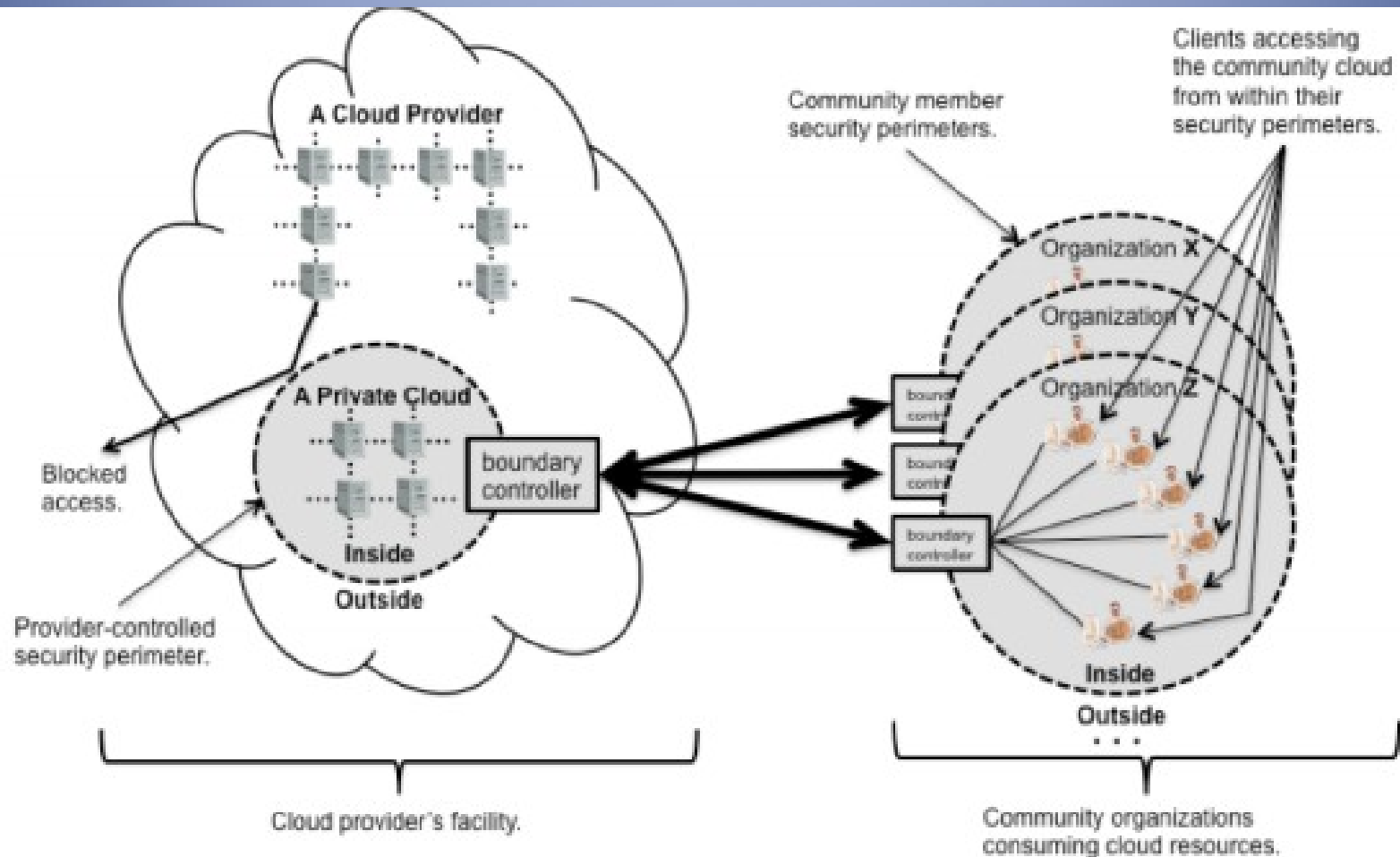


Figure 6: Outsourced Community Cloud

Cloud Types

- Deployment models.
- Public cloud.
 - The cloud infrastructure is made available to the general public
 - May be used by large industry group
 - Generally owned by an organization selling cloud services.
- Public clouds are run by third parties, and jobs from many different customers may be mixed together on the servers, storage systems, and other infrastructure within the cloud.
- End users don't know who else's job may be running on the same server, network, or disk as their own jobs.

Public Cloud

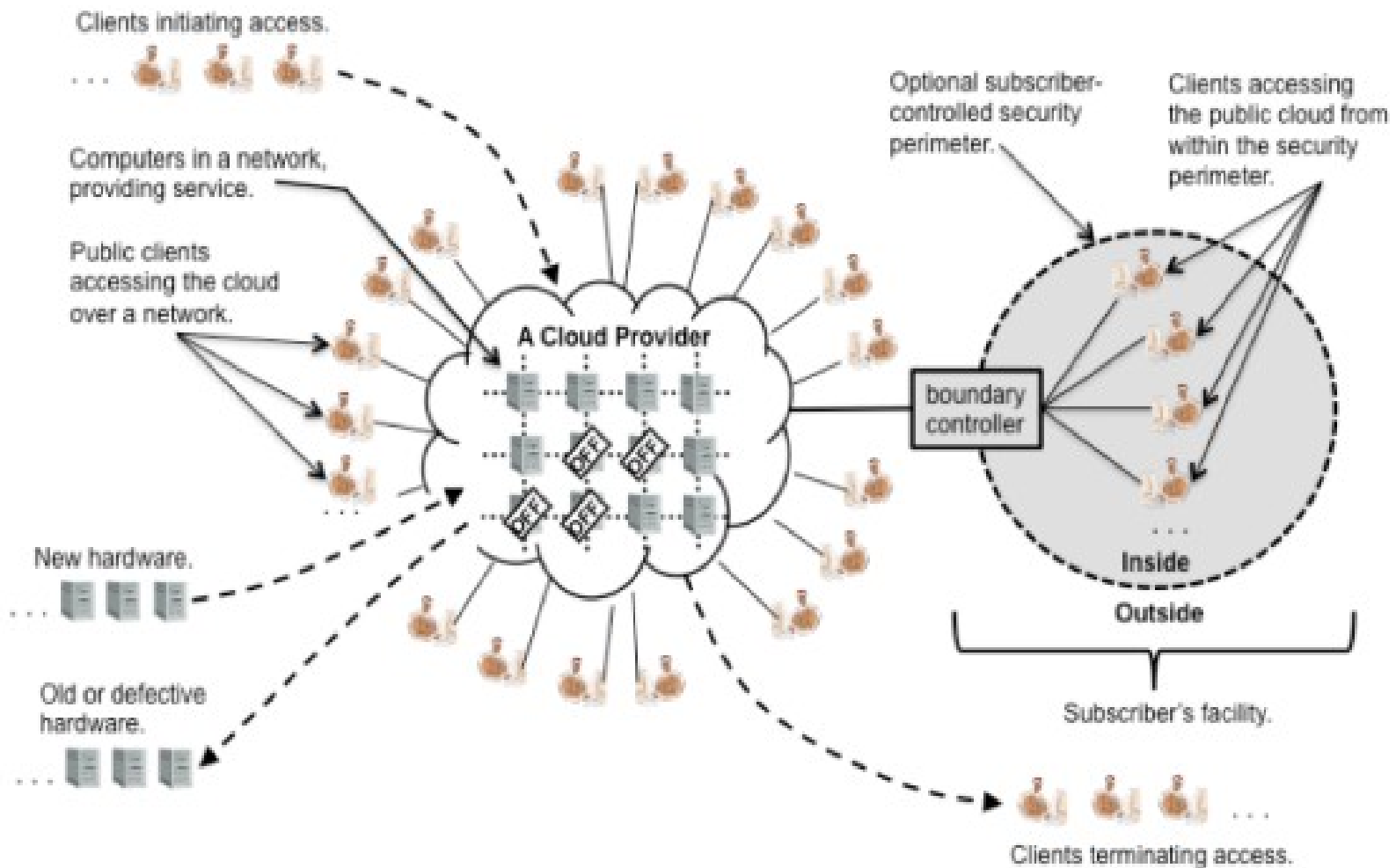


Figure 7: Public Cloud

Cloud Types

- Deployment models.
- Hybrid cloud
 - The cloud infrastructure is a composition of two or more clouds (private, community, or public)
 - bound together by standardized or proprietary technology that enables data and application portability (e.g., cloud bursting for load-balancing between clouds).

Cloud Types

- Service models.
- Software as a Service (SaaS).
- SaaS is at the highest layer and features a complete application offered as a service, on-demand, via multi-tenancy ,meaning a single instance of the software runs on the provider's infrastructure and serves multiple client organizations.

Cloud Types

- Service models.
- Software as Service (SaaS).
 - The service provided to the consumer is to use the provider's applications running on a cloud infrastructure.
 - The applications are accessible from various client devices
 - Generally a thin client interface such as a Web browser is provided.
 - The consumer does not manage or control the underlying cloud infrastructure including network, servers, operating systems, storage, or even individual application capabilities,
 - exception of limited user-specific application configuration settings.

Software as a Service

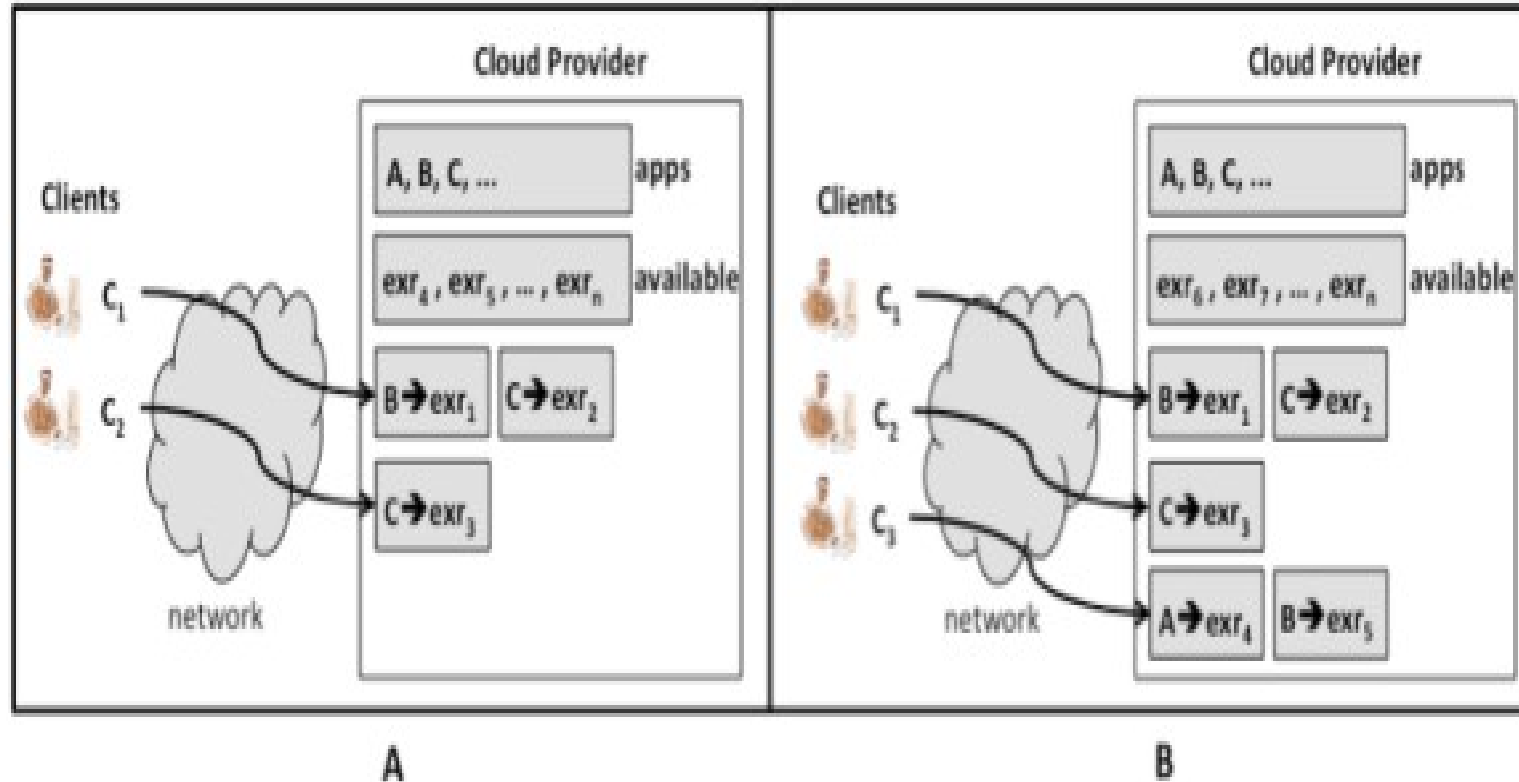


Figure 9: SaaS Subscriber/Provider Interaction Dynamics

Scope of Control

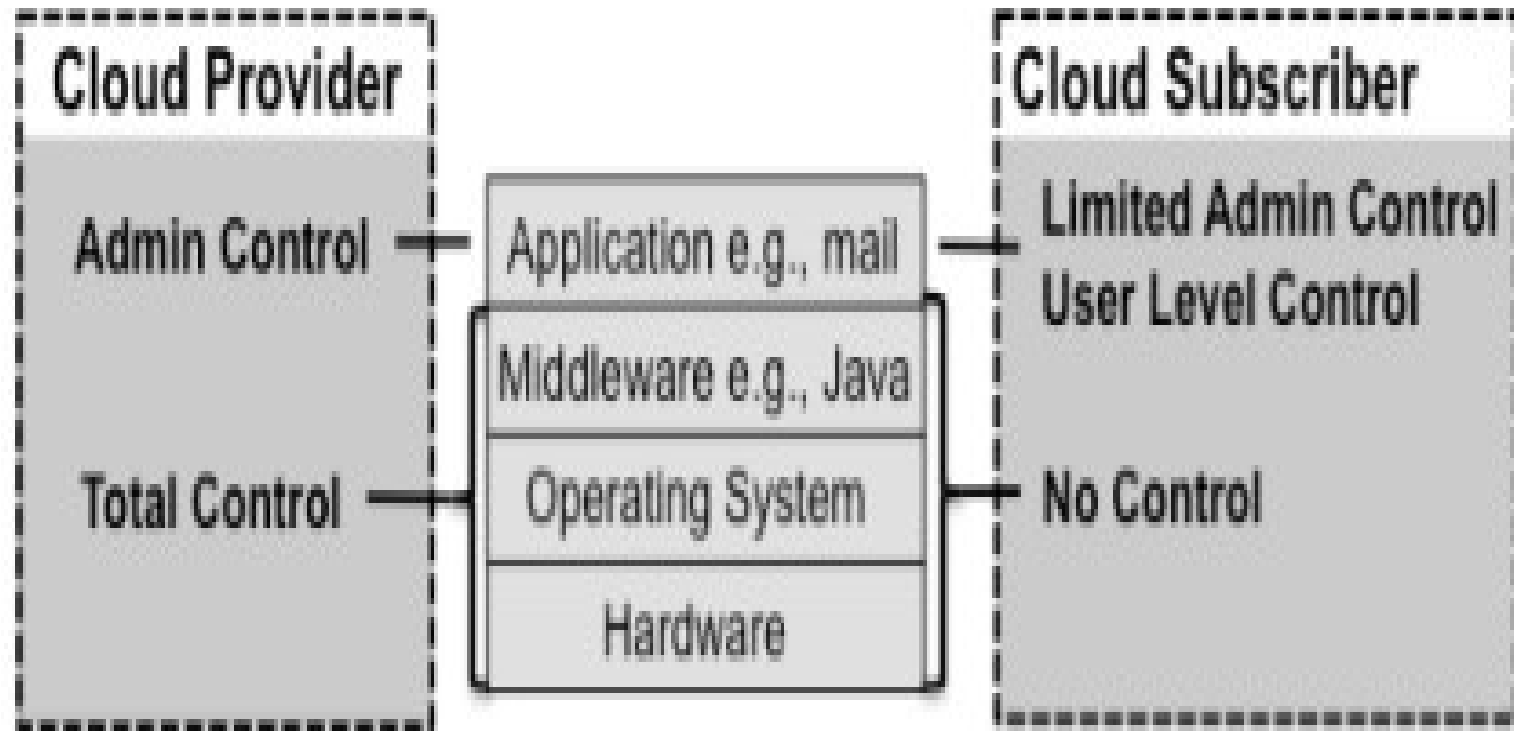


Figure 10: SaaS Provider/Subscriber Scope of Control

Cloud Providers

- SaaS –
 - Google Apps
 - Microsoft Office 365
 - Zoho office
 - Salesforce.com
 - SQL Azure
 - Oracle on Demand

Software as a Service

- LiveOps

- LiveOps Contact Center Application Suite, which integrates call-center functions such as chat and e-mail, inbound call routing, interactive voice response and workforce management
- <http://www.liveops.com/>

- Knowledge Tree

- KnowledgeTree enables traditional document management and collaboration features, including document versioning and auditing, metadata and content searching, workflow, tagging and tag clouds, RSS feeds and e-mail triggers. Useful for sales teams.
- <https://www.knowledgetree.com/>

Software as a service

- Taleo
 - Taleo operates the Talent Management Cloud, which it says comprises scalable, elastic and secure infrastructure; an open, mobile and flexible software platform and a full suite of talent management applications.
 - These include recruiting, performance management, compensation management, employee development and succession planning.
 - [Taleo Website](#)

Software as a Service

- Project Management Software Providers
- Project managers have a number of cloud-based options to manage projects, which can help streamline their tasks and improve collaboration.
 - MS Project
 - Clarizen
 - GroupCamp
 - Planzone
 - Zoho Projects.

Software as a Service

- Help Desk Software Providers
- SaaS help desk solutions can streamline support administration, reduce infrastructure costs, and improve the way you communicate with your internal and external customers by extending support to different types of users.
 - itBit
 - FreshDesk
 - ZenDesk
 - Desk.com

Software as a Service

- PIXLR
- It is a cloud based collection of photo editing tools.
- Pixlr was founded in Sweden in 2008 by Ola Sevandersson. On 19 July 2011, the suite was acquired by Autodesk.
- <https://pixlr.com/>

Software as a Service

- **QuickBooks**

- QuickBooks is a very simple and easy to use online accounting software which helps you manage business finances in just a click
- <http://www.quickbooks.in/features/>

- **BlueJeans Network**

- It is a company that provides cloud-based video communications service.
- Every BlueJeans member is provided with a private “meeting room” in the BlueJeans cloud to schedule and host conference meetings. It brings together business conferencing solutions like Cisco, Microsoft Lync, Lifesize, and Polycom with consumer services like Google.

Cloud Types

- Service models.
- Platform as Service (PaaS).
 - Consumer uses cloud to deploy his applications.
 - Applications can be consumer-created or acquired.
 - applications created using programming languages and tools supported by the provider.
 - The consumer does not manage or control the underlying cloud infrastructure
 - Has control over the deployed applications and possibly application hosting environment configurations.

Platform as a Service

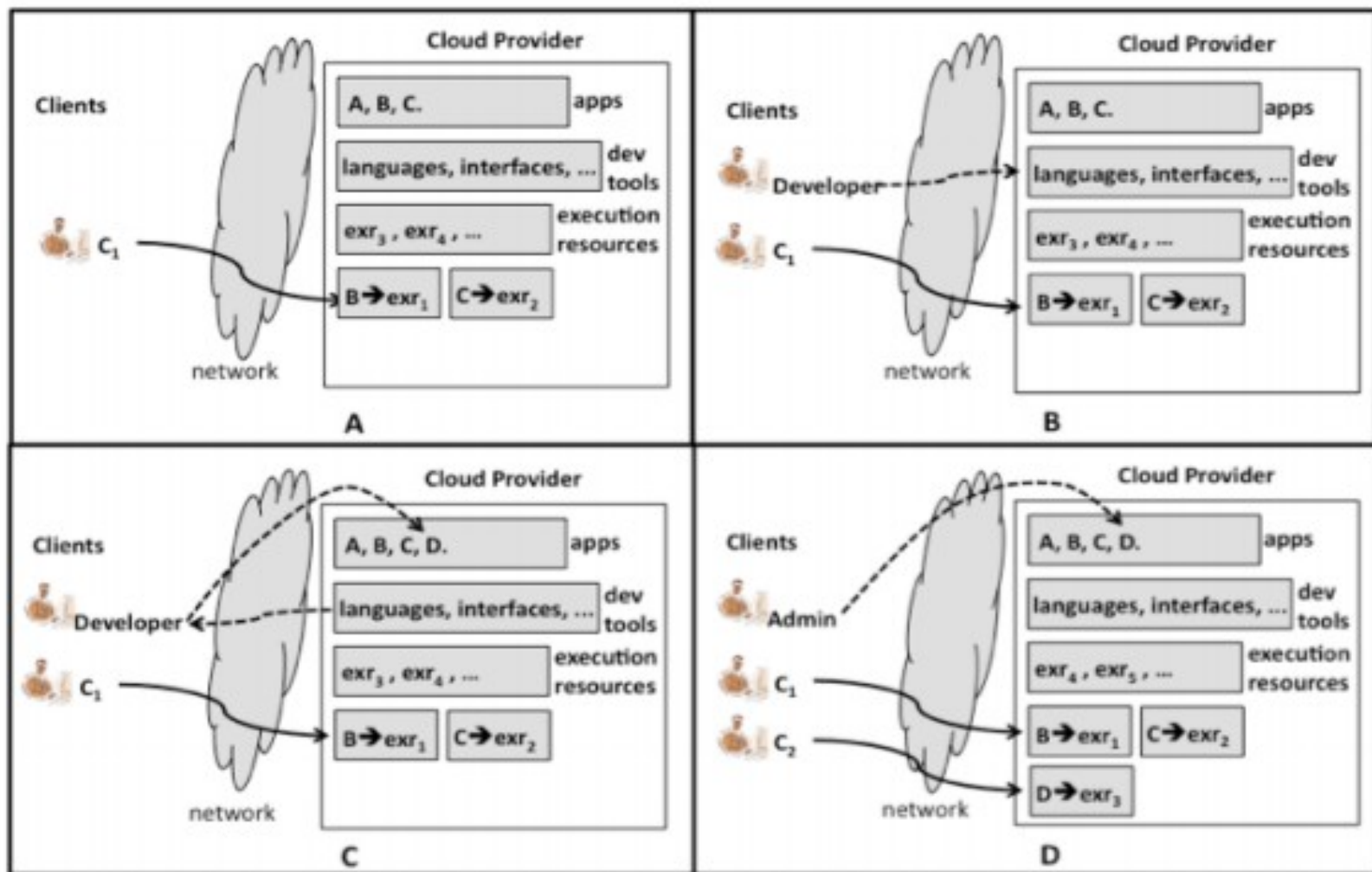


Figure 13: PaaS Subscriber/Provider Interaction Dynamics

Scope of Control

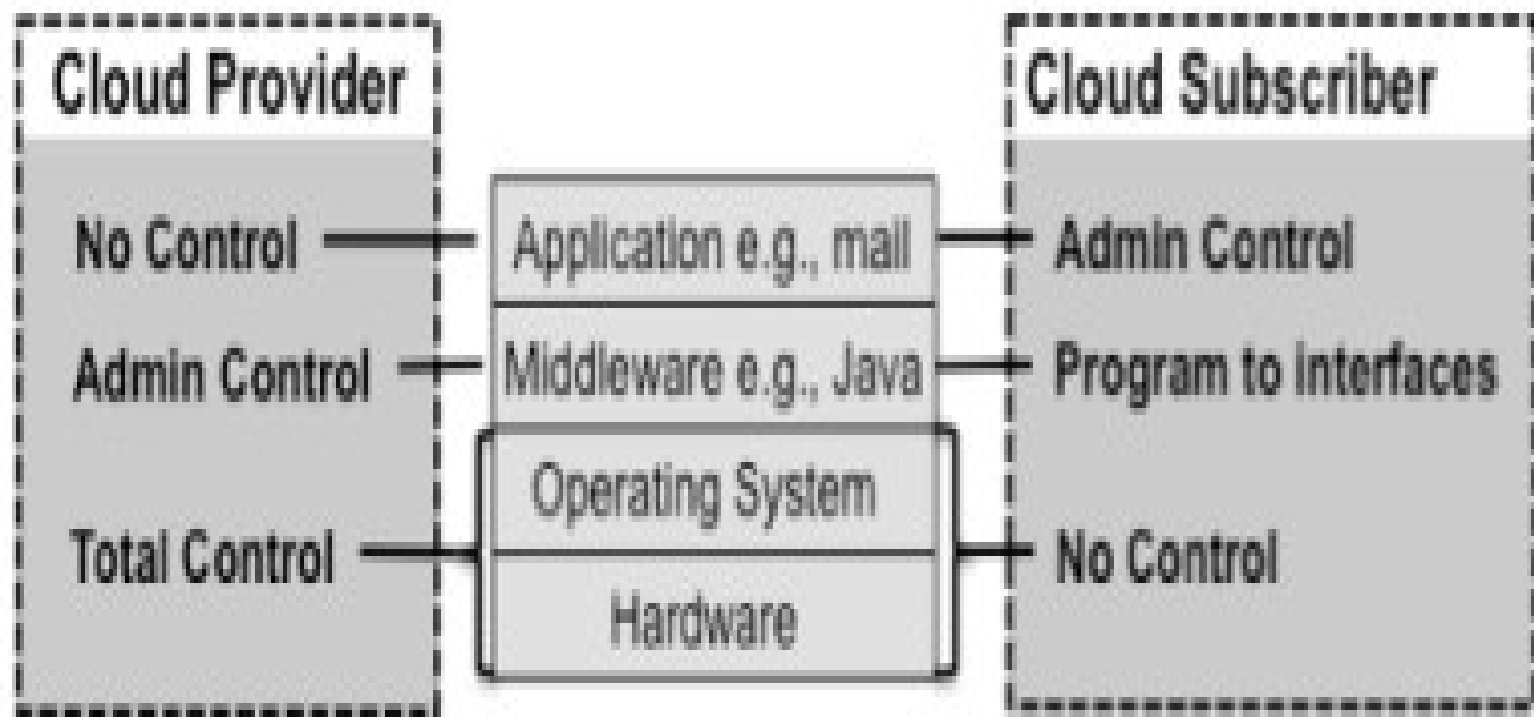


Figure 14: PaaS Component Stack and Scope of Control

Cloud Providers

- PaaS –
 - Google AppsEngine
 - Microsoft Azure
 - Force.com
 - GoGrid CloudCenter
 - Amazon Web Services
 - Long Jump
 - IBM Smart Cloud

Platform as a Service

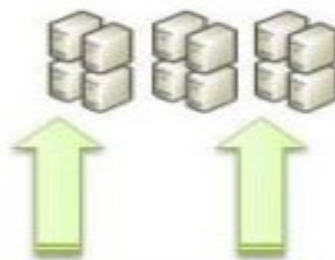
- Engine Yard
 - Engine Yard is designed for web application developers using Ruby on Rails, PHP and Node.js who want to take advantage of cloud computing without the operations management responsibility.
 - Engine Yard provides a set of services on top of Amazon AWS.

Platform as a Service

- Red Hat OpenShift
 - Red Hat OpenShift is based on open source applications and offers a wide variety of languages, databases and components.
 - The PaaS is highly customizable and offered in three forms: OpenShift Online (a cloud-based hosting service), OpenShift Enterprise (a private PaaS that runs in your data center) and OpenShift Origin (the open source application hosting platform).
 - OpenShift automates system administration tasks such as virtual server provisioning, configuration and scaling and supports git repositories for code management.

Platform as a Service

- APPFOG
 - AppFog is a multi-language, multi-framework PaaS that's a good option for multi-cloud deployments, including private clouds.
 - AppFog supports Java, Ruby, PHP, Python, Node, Scala and Erlang and offers MySQL, PostgreSQL, Redis and RabbitMQ along with third party add-on services.
 - It can be used as a Database as a Service with ClearDB, MongoHQ, MongoLab, Redis Cloud and Xeround.



Developers

Cloud Types

- Service models.
- Infrastructure as Service (IaaS).
 - The processing, storage, networks, and other fundamental computing resources are provided by the cloud to the consumer.
 - the consumer is able to deploy and run arbitrary software, which can include operating systems and applications.
 - The consumer does not manage or control the underlying cloud infrastructure but has control over operating systems, storage, deployed applications, and possibly limited control of select networking components (e.g., host firewalls).

Infrastructure as a Service

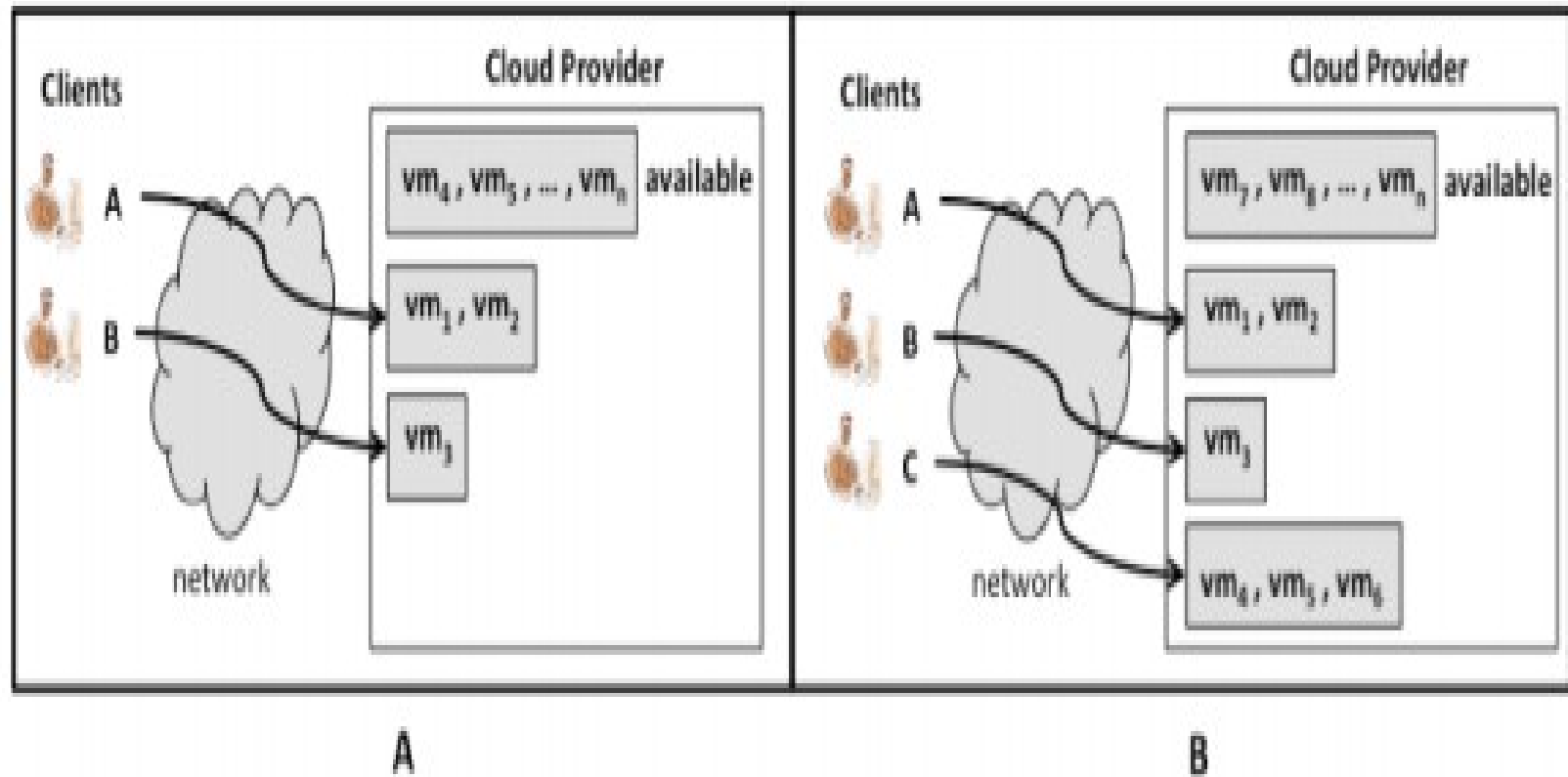


Figure 15: IaaS Provider/Subscriber Interaction Dynamics

Scope of Control

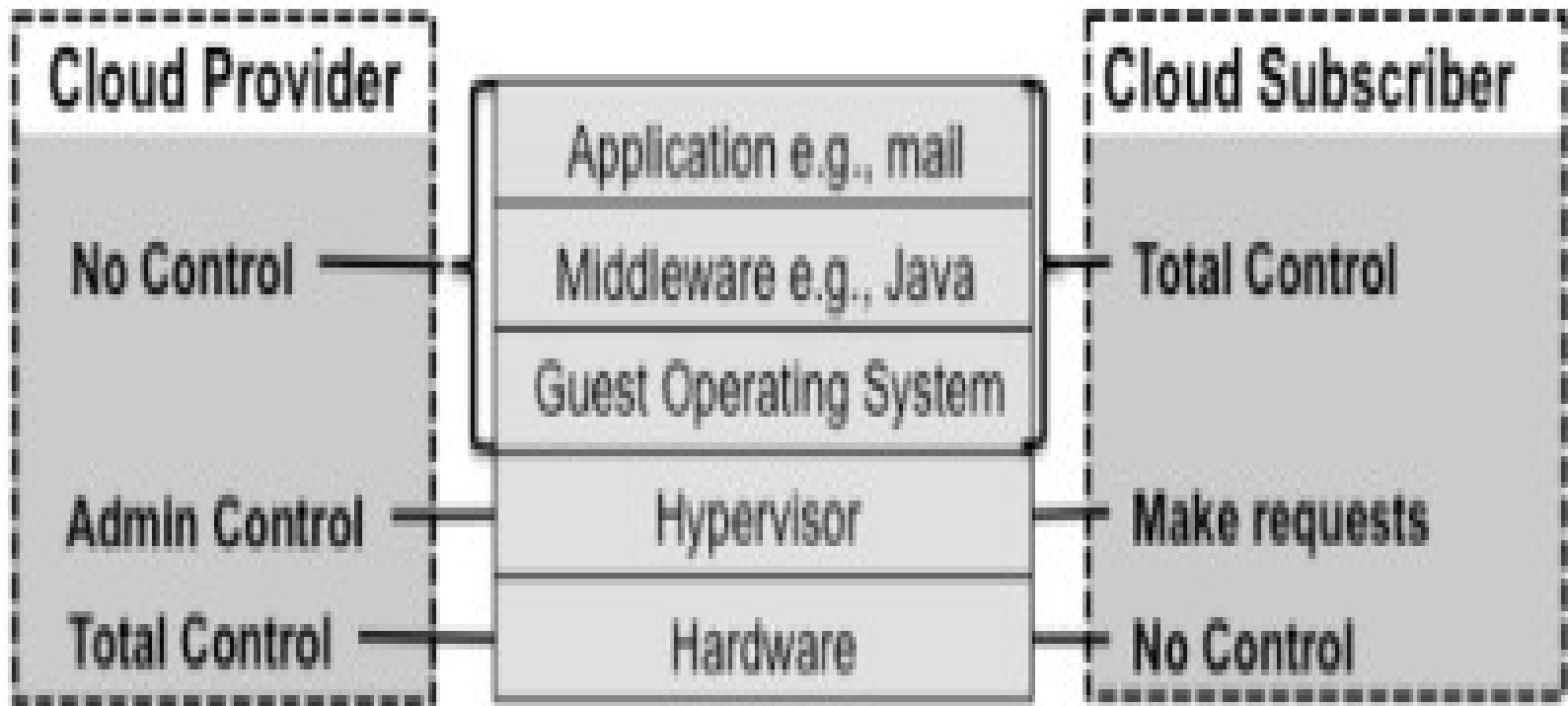


Figure 16: IaaS Component Stack and Scope of Control

Cloud Providers

- IaaS –
 - Eucalyptus
 - Windows Azure
 - GoGrid
 - Rackspace Cloud
 - DeskTone
 - CloudSigma

Infrastructure as a Service

- Amazon Web Services
 - Amazon Web Services offers a full range of compute and storage offerings, including on-demand instances and specialized services such as Amazon Elastic Map Reduce (EMR) and Cluster GPU instances, as well as Elastic Block Storage (EBS) and high performance SSDs on the storage side.
 - Additionally, the IaaS offers infrastructure services such as workflows, message passing, archival storage, in-memory caching services, search services, both relational and NoSQL databases and more.

Infrastructure as a service

- IBM Smart Cloud Enterprise
 - The IaaS is ideal for enterprises managing a large number of developers and testers who need to deploy virtual machines and allocate storage as efficiently as possible.
 - You can manage administrator and user roles, set limits on resources users can deploy and readily report on user activity.

Infrastructure as a Service

- HP Enterprise Converged Infrastructure
 - HP's cloud is built on OpenStack and its IaaS service is part of the company's Converged Cloud Solutions for public, hybrid and private clouds.
 - HP offers Windows and Linux command line interfaces in addition to the dashboard as well as a RESTful API so you can use cURL for low level access to HP's cloud functions.
 - HP cloud also offers simple access to its content distribution network (CDN), particularly useful for companies with a global customer and/or user base.

Cloud -Essential Characteristics

Broad
Network Access

Rapid Elasticity

Measured Service

On-Demand
Self-Service

Resource Pooling

*Essential
Characteristics*

Software as a
Service (SaaS)

Platform as a
Service (PaaS)

Infrastructure as a
Service (IaaS)

*Service
Models*

Public

Private

Hybrid

Community

*Deployment
Models*

Essential Characteristics

- On-demand self-service.
 - A consumer should receive computing capabilities, such as server time and network storage, as needed automatically without requiring human interaction with each service's provider.
- Broad network access.
 - services are available over the network and accessed through standard mechanisms
 - Support for use of heterogeneous thin or thick client platforms (e.g., mobile phones, laptops, and personal digital assistants (PDAs)).

Essential Characteristics

- Resource pooling.
 - Support for computing resources pooling to serve multiple consumers using a multi-tenant model
 - different physical and virtual resources should be dynamically assigned and reassigned according to consumer demand.
 - There is a sense of location independence in that the subscriber generally has no control or knowledge over the exact location of the provided resources but may be able to specify location at a higher level of abstraction (e.g., country, state, or datacenter).

Essential Characteristics

- Rapid elasticity.
 - Services or resources can be rapidly and elastically provisioned, in some cases automatically, to quickly scale out and rapidly released to quickly scale in.
 - To the consumer, the capabilities available for provisioning often appear to be unlimited and can be purchased in any quantity at any time.

Essential Characteristics

- Measured Service.
 - Cloud systems automatically control and optimize resource use by leveraging a metering capability at some level of abstraction appropriate to the type of service (e.g., storage, processing, bandwidth, and active user accounts).
 - Resource usage can be monitored, controlled, and reported providing transparency for both the provider and consumer of the utilized service.

Cloud Benefits

Benefits of Cloud Computing

- IT Efficiency — Minimize costs where companies are converting their IT costs from capital expenses to operating expenses through technologies such as virtualization.
- Cloud computing begins as a way to improve infrastructure resource deployment and utilization, but fully exploiting this infrastructure eventually leads to a new application development model.

- Netflix, is taking advantage of cloud resources as it meets up and down demand for its Internet subscription service for movies and TV shows.
- “Because it streams many movies and shows on demand, the company faces large surges of capacity at peak times,” the report explains. “

- As Netflix began to outgrow its data center capabilities, the company made a decision to migrate its Website and streaming service from a traditional data center implementation to a cloud environment.
- This move allowed the company to grow and expand its customer base without having to build and support a data center footprint to meet its growth requirements.”

Cloud Success Stories

- In 2013, **Novartis** ran a project that involved virtually screening 10 million compounds against a common cancer target in less than a week. They calculated that it would take 50,000 cores and close to a \$40 million investment if they wanted to run the experiment internally. Partnering with Cycle Computing and **Amazon Web Services (AWS)**, Novartis built a platform leveraging **Amazon Simple Storage Service (Amazon S3)**, **Amazon Elastic Block Store (Amazon EBS)**, and four Availability Zones. The project ran across 10,600 Spot Instances (approximately 87,000 compute cores) and allowed Novartis to conduct 39 years of computational chemistry in 9 hours for a cost of \$4,232. Out of the 10 million compounds screened, three were successfully identified.

Cloud Success Stories

- Pinterest is no stranger to rapid growth, expanding from 50,000 users to 17 million in 9 months. Now at 48 million users, Pinterest was able to scale its business because it was built on Amazon Web Services (AWS). With a company of fewer than 12 employees, Pinterest didn't want to dedicate staff time to managing a data center. Instead, Pinterest uses AWS to manage a high-performance social application that stores more than 8 billion objects and 400 terabytes of data in the AWS Cloud using **Amazon Simple Storage Service** (Amazon S3), and 225,000 instance hours a month with Amazon Elastic Compute Cloud (**Amazon EC2**).

Benefits of Cloud Computing

- *“We really don’t want to operate datacenters anymore. We’d rather spend our time giving our customers great service and writing great software than managing physical hardware.”*

—Don MacAskill, CEO, SmugMug

Benefits of Cloud Computing

- Eliminate over provisioning — Cloud computing provides scaling on demand, which, when combined with utility pricing, removes the need to overprovision to meet demand.
- With cloud computing, companies can scale up to massive capacities in an instant.

Benefits of Cloud Computing

- Accelerated cycles — The cloud computing model provides a faster, more efficient way to develop the new generation of applications and services.
- Faster development and testing cycles means businesses can accomplish in hours what used to take days, weeks, or months.

Benefits of Cloud Computing

- Increase agility — Cloud computing accommodates change like no other model.
- Animoto Productions, makers of a mashup tool that creates video from images and music, used cloud computing to scale up from 50 servers to 3,500 in just three days.
- Cloud computing can also provide a wider selection of more lightweight and agile development tools, simplifying and speeding up the development process.

Benefits of Cloud Computing

- For end users, cloud computing means there are no hardware acquisition costs, no software licenses or upgrades to manage, no new employees or consultants to hire, no facilities to lease, no capital costs of any kind — and no hidden costs.
- Just a metered, per-use rate or a fixed subscription fee.
- Use only what you want, pay only for what you use.

Benefits of Cloud Computing

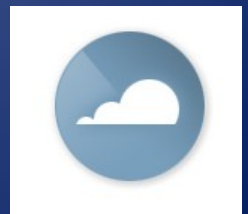
- Lower Costs
 - Usage based billing.
 - Supports multi-tenancy
 - Reduction in operational costs
 - Minimal infrastructure Up gradation costs
 - Reduced Licensing Fees
- Ease of utilization
 - Browser based access
 - Location independent access
- Quality of Service
 - On demand from provider

Benefits of Cloud Computing

- Reliability
 - The scale of cloud computing networks and their ability to provide load balancing and failover.
- Outsourced IT management
 - Cloud provider manages IT infrastructure
 - Control over IT expenses
- Simplified Maintenance and Upgrade
 - Centralized System
 - Easy patching and upgrade
- Low Barrier to Entry

Q&A

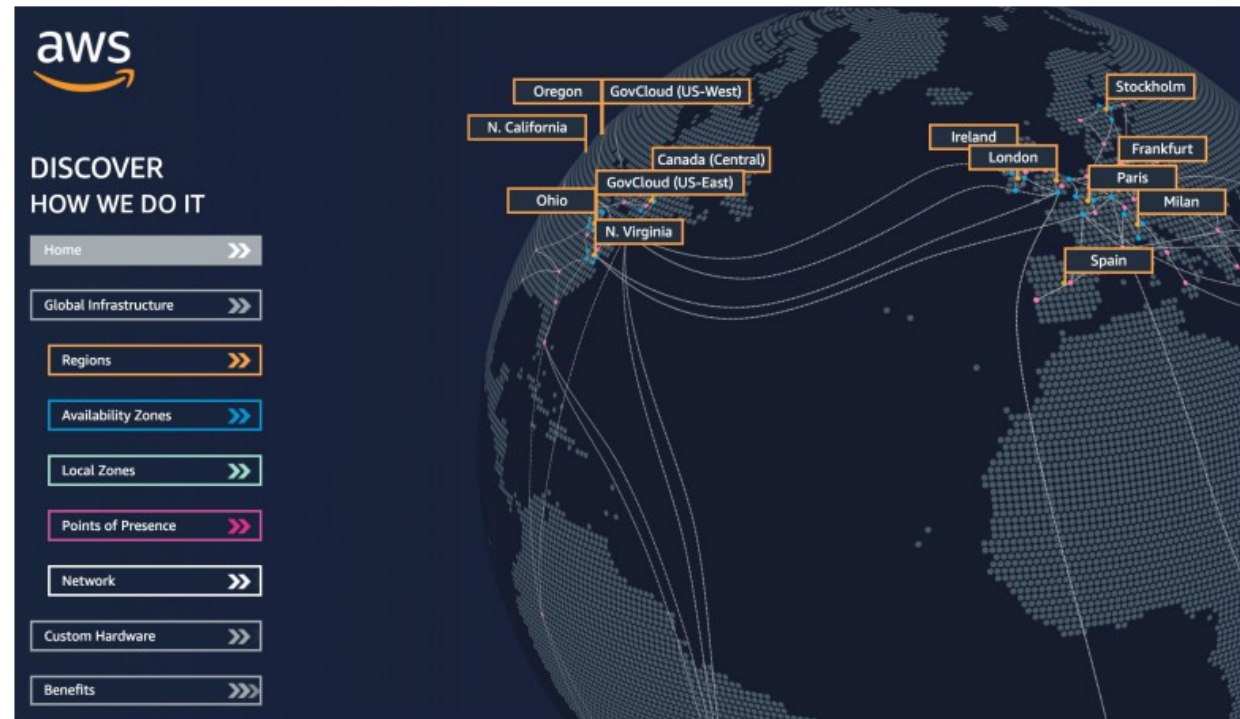
Thanks



AWS Introduction

AWS Global Infrastructure

- AWS Regions
- AWS Availability Zones
- AWS Data Centers
- AWS Edge Locations / Points of Presence
- <https://infrastructure.aws/>



AWS Regions

AWS Regions

- AWS has **Regions** all around the world
- Names can be us-east-1, eu-west-3...
- A region is a **cluster of data centers**
- Most AWS services are region-scoped



<https://aws.amazon.com/about-aws/global-infrastructure/>

US East (N. Virginia) us-east-1

US East (Ohio) us-east-2

US West (N. California) us-west-1

US West (Oregon) us-west-2

Africa (Cape Town) af-south-1

Asia Pacific (Hong Kong) ap-east-1

Asia Pacific (Mumbai) ap-south-1

Asia Pacific (Seoul) ap-northeast-2

Asia Pacific (Singapore) ap-southeast-1

Asia Pacific (Sydney) ap-southeast-2

Asia Pacific (Tokyo) ap-northeast-1

Canada (Central) ca-central-1

Europe (Frankfurt) eu-central-1

Europe (Ireland) eu-west-1

Europe (London) eu-west-2

Europe (Paris) eu-west-3

Europe (Stockholm) eu-north-1

Middle East (Bahrain) me-south-1

South America (São Paulo) sa-east-1

How to choose an AWS Region?

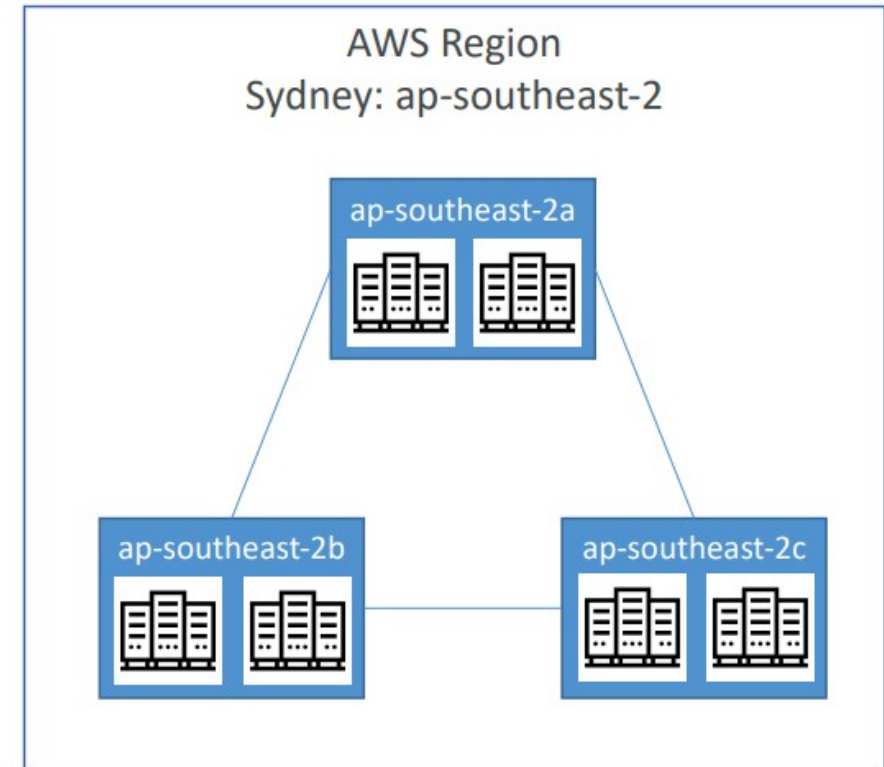
If you need to launch a new application, where should you do it?



- **Compliance** with data governance and legal requirements: data never leaves a region without your explicit permission
- **Proximity** to customers: reduced latency
- **Available services** within a Region: new services and new features aren't available in every Region
- **Pricing**: pricing varies region to region and is transparent in the service pricing page

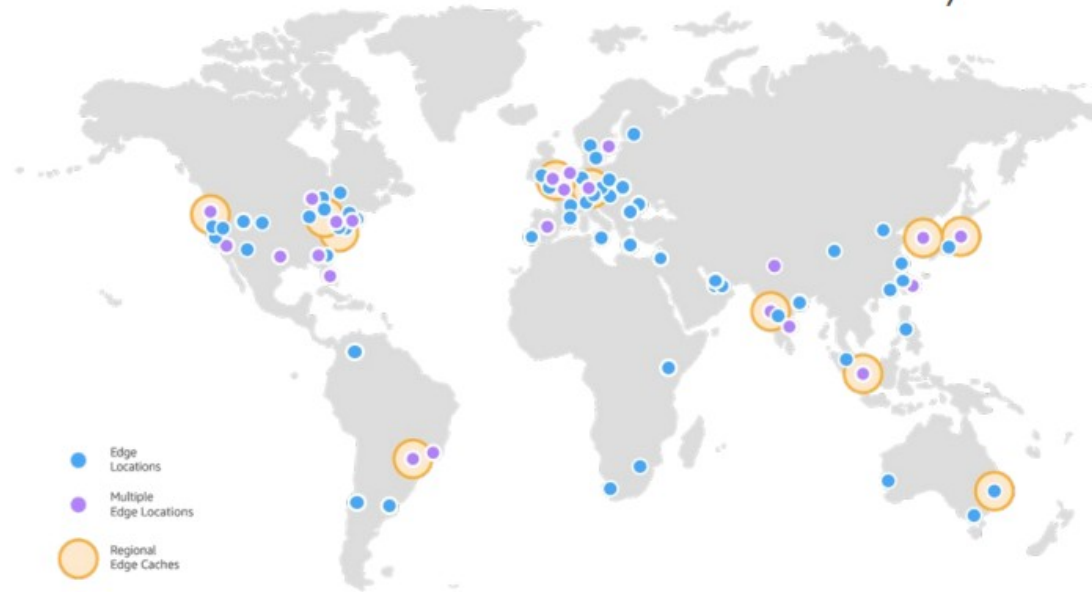
AWS Availability Zones

- Each region has many availability zones (usually 3, min is 2, max is 6). Example:
 - ap-southeast-2a
 - ap-southeast-2b
 - ap-southeast-2c
- Each availability zone (AZ) is one or more discrete data centers with redundant power, networking, and connectivity
- They're separate from each other, so that they're isolated from disasters
- They're connected with high bandwidth, ultra-low latency networking



AWS Points of Presence (Edge Locations)

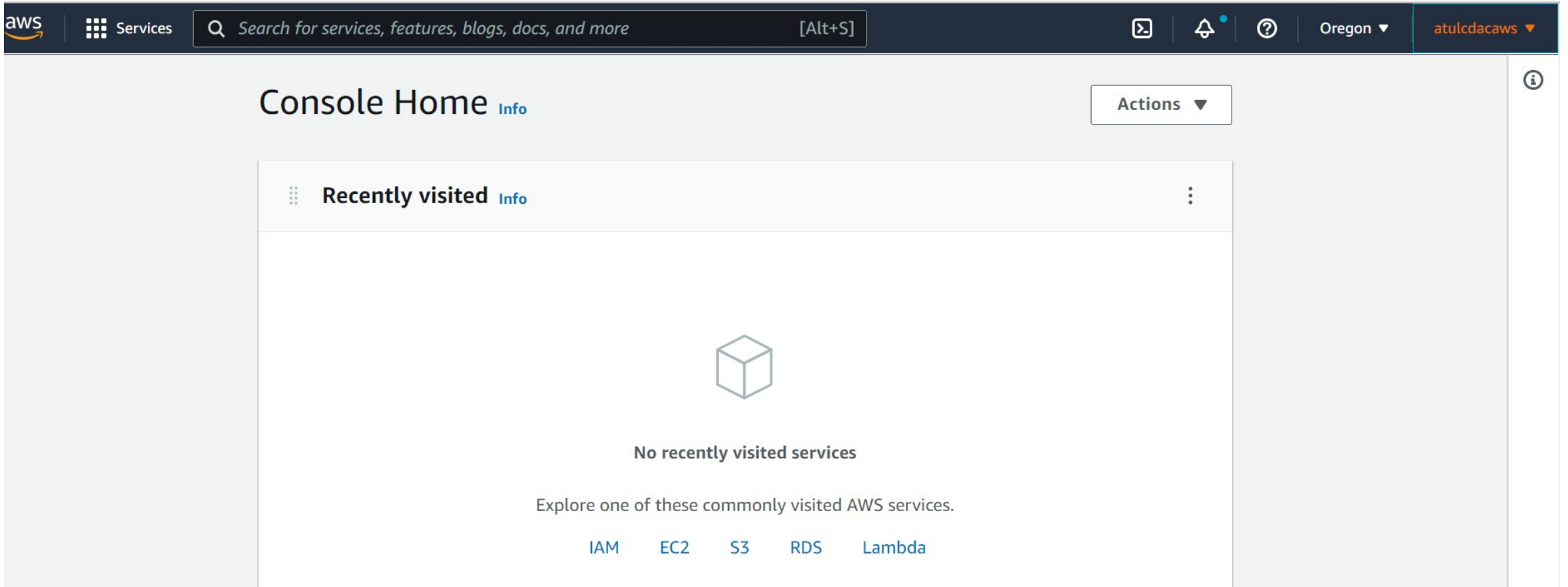
- Amazon has 216 Points of Presence (205 Edge Locations & 11 Regional Caches) in 84 cities across 42 countries
- Content is delivered to end users with lower latency



Signup

- Signup for AWS with your email id
- Remember the id, password and also the AWS account id (different from your email id) – This creates a **root account**
- You will require your phone and credit card

After successful signup



Set Up Billing Alerts – Billing and Cost Management

- Search for *Billing and Cost Management*
- In its menu, scroll down and click on *Billing Preferences*
- Edit *Invoice delivery preferences* and check the check box against *PDF invoices delivered by email* and then *Update*
- Under *Alert preferences*, check the check box against *Receive AWS Free Tier Alerts* and type the email where we want to receive these
- Check the checkbox against *Receive CloudWatch billing alerts* and then click *Update*

Set Up Billing Alerts- CloudWatch

- Now search for *CloudWatch*
- Change region to *US East (N. Virginia) us-east-1*, since all billing information is located here
- Click Alarms->Billing->Create alarm->Select metric
- Under Metric, click on Billing->Total Estimated Charge->Check USD->Select metric->Leave everything default as is but change amount to 5->Next
- Select the radio button *Create new topic*->In the email id, enter your email id->Create topic->Next
- Give some alarm name->Next
- Create alarm

EC2

EC2 Basics

- One of the most popular AWS offerings
- EC2 = Elastic Compute Cloud = IaaS
- Capabilities:
 - Rent a VM (EC2)
 - Store data on a virtual drive (EBS)
 - Distribute load across machines (ELB)
 - Scale the services using auto scaling group (ASG)

EC2 User Data

- **EC2 User Data** runs a user-defined script only once at the first start (instance creation)
- Used to automate boot tasks such as
 - Installing updates
 - Installing software
 - Downloading common files from the internet
 - Etc
- EC2 User Data script must be run as a root user, so we may need *sudo*
- Note: If we want to run this every time, we need to put the script in */var/lib/cloud/scripts/per-boot/*

Create an EC2 Instance using EC2 User Data

- Search for EC2, then click on Instances, Launch Instances
- Leave everything as default, but click on *Create new key pair*, as it is required – Give some name, and select defaults
- Under Network, select *Allow Http traffic from the internet*
- In *Advanced details-User data*, enter the following script
 - `#!/bin/bash`
 - `# Use this for your user data (script from top to bottom)`
 - `# install httpd (Linux 2 version)`
 - `yum update -y`
 - `yum install -y httpd`
 - `systemctl start httpd`
 - `systemctl enable httpd`
 - `echo "<h1>Hello World from $(hostname -f)</h1>" > /var/www/html/index.html`
- Launch instance

Test the EC2 Instance

- When the instance is running, open its public IP address in the browser

AWS S3

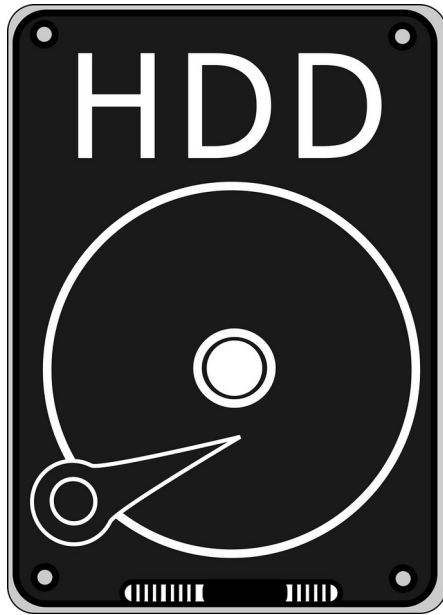
Block Storage

File Storage

Object Storage

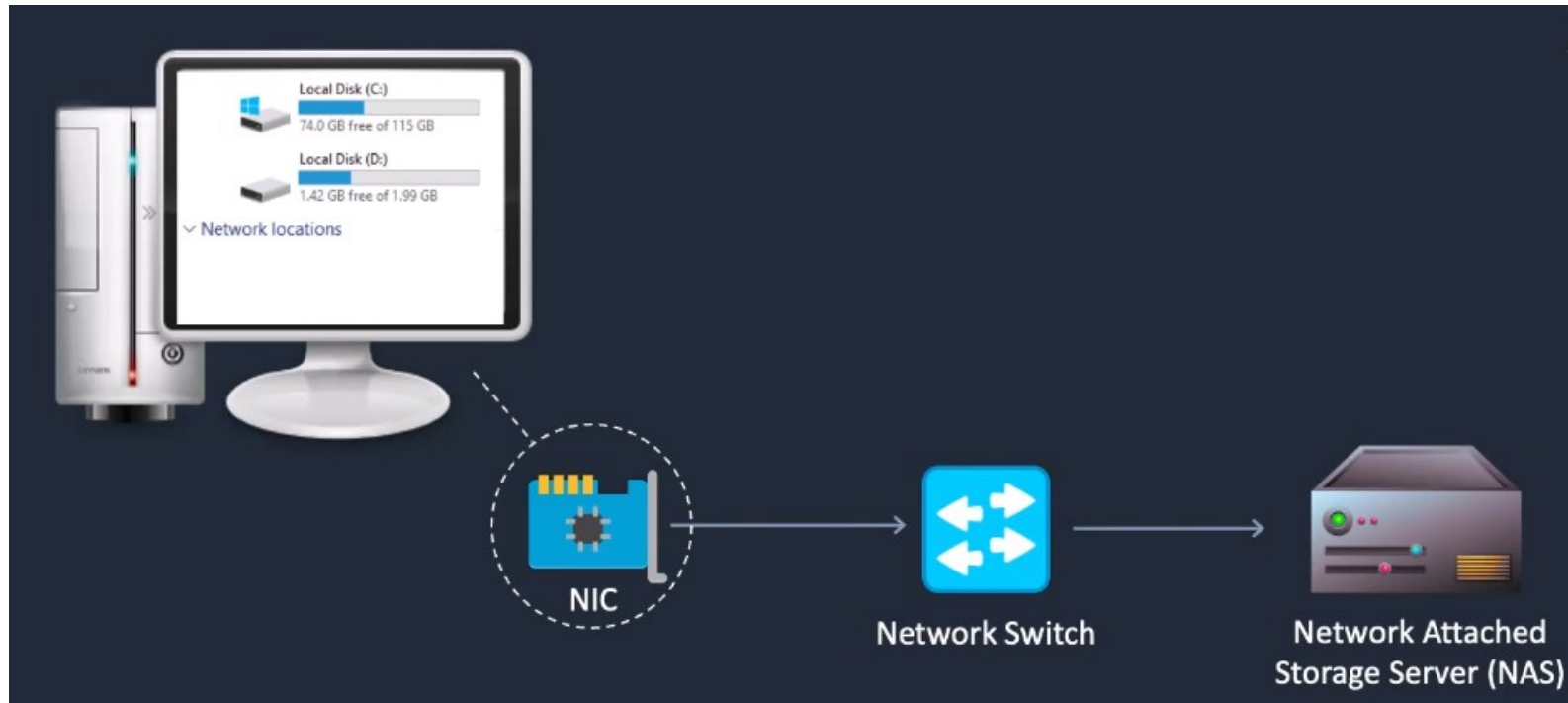
AWS Storage Options: Block Storage

- **Block storage:** Hard disk drive storage (HDD/SSD)



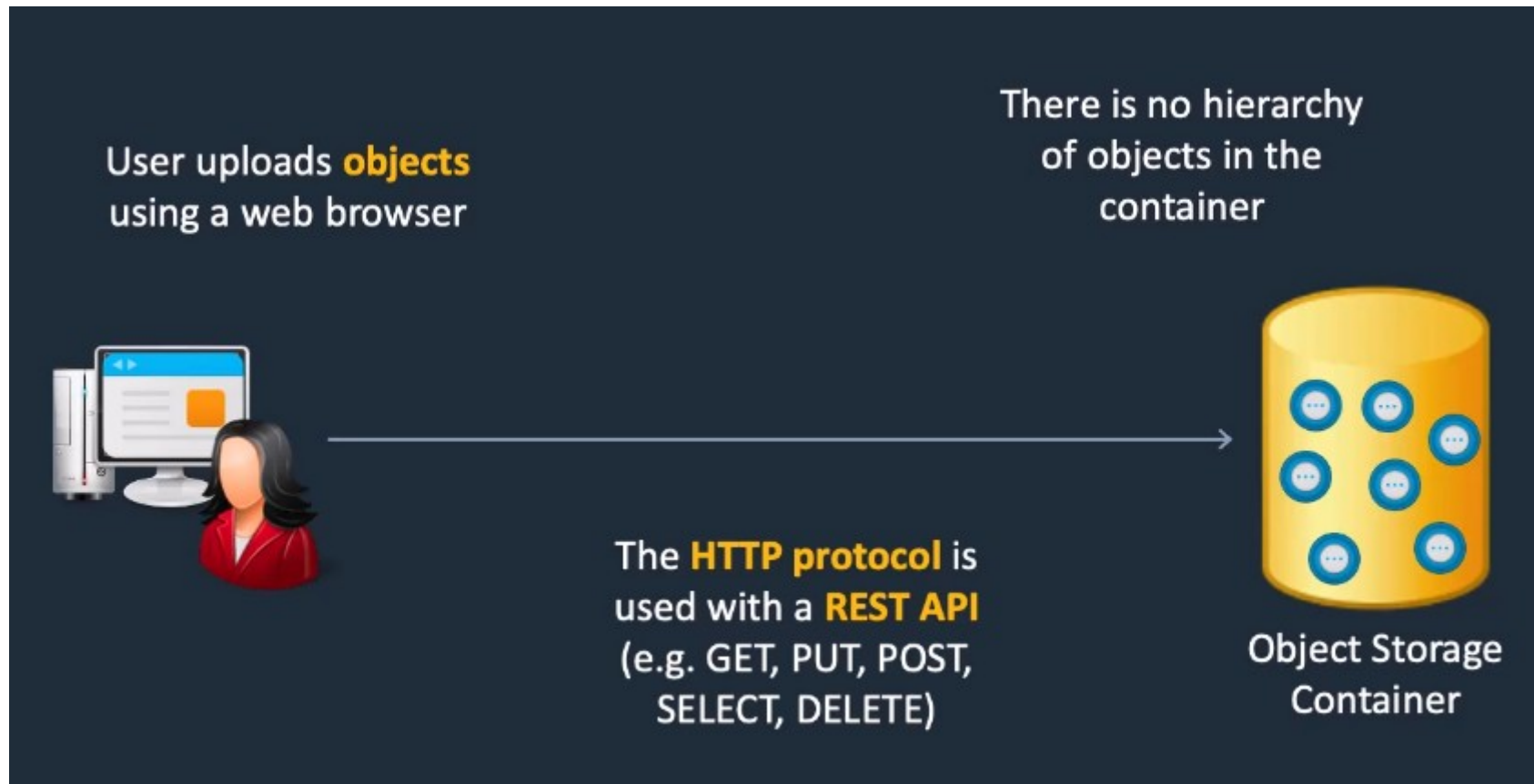
AWS Storage Options: File Storage

- **File storage:** Allows access to a block storage system over a network



AWS Storage Options: Object Storage

- **Object storage:** Allows storage and retrieval of files as if they are objects



Simple Storage Service (S3): Object Storage in AWS

- **Bucket:** Collection of **objects**



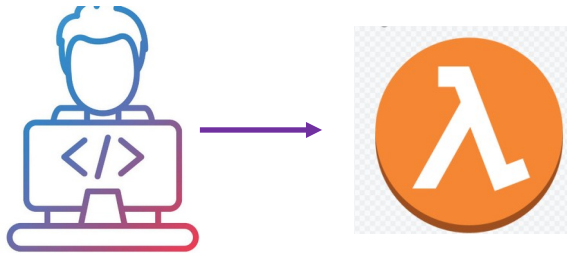
S3: Object Storage in AWS

- Search for S3->Create bucket->Bucket name: *Must be globally unique*->Leave all defaults->Create bucket
- Click on the bucket name->Upload->Upload 1-2 files from your disk->Upload

Serverless and AWS Lambda

Lambda and Serverless Computing

- **Serverless computing:** Does not mean *no server*, But it means that *the developers do not have to worry about servers*
- Developers just need to deploy code, AWS will manage the servers
- **AWS Lambda:** Event-driven, serverless computing platform

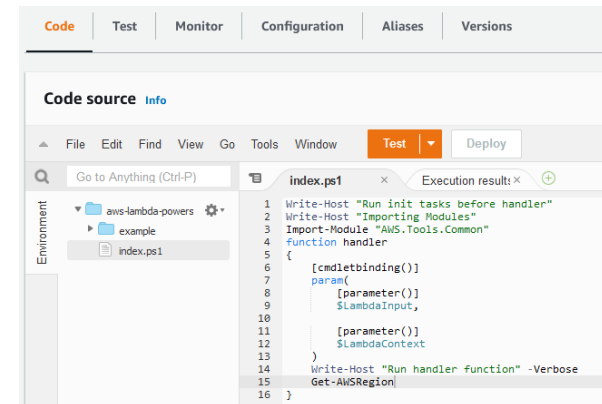


Developer creates and deploys a Lambda Function

```
import json
import logging
import boto3

# Initialize logging
logger = logging.getLogger()
logger.setLevel(logging.INFO)

def lambda_handler(event, context):
    # Log the raw event
    logger.info("Event: " + json.dumps(event))
```



An event (e.g. file upload in S3) triggers it

AWS Lambda

- We need not worry about which AWS resources to launch, or how we will manage them
- Instead, we just put our code on Lambda, and it will run
- The code is executed based on the response of events in AWS services such as add/delete files in S3 bucket, HTTP request from Amazon API Gateway, etc
- Can be used only for executing background tasks

Lambda Overview

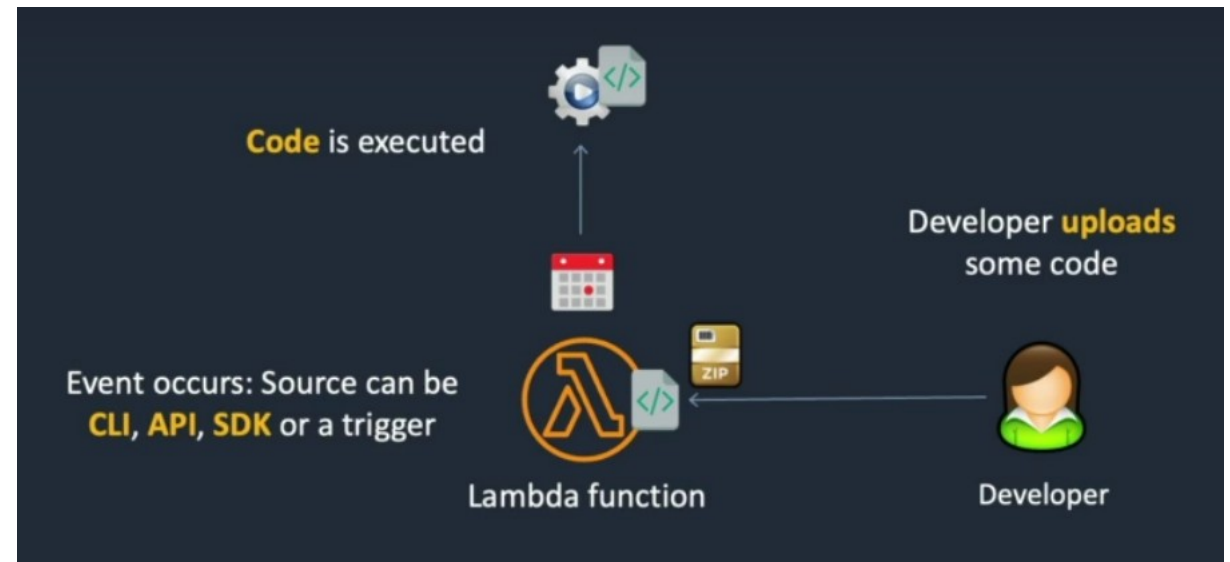
- **AWS Lambda: A serverless compute service**



- Developer uploads some code – no payment at this stage
- Charges only apply to the *usage* of the code

Lambda Overview

- An event occurs
- Triggers the Lambda function
- It can run for max 15 mins



Creating Lambda Function

- Search for Lambda->Create function->Author from scratch->Function name:my-lambda-s3-function->Runtime:Python 3.11->Create function
- Code: Remove the existing code and copy and paste the code from the next slide->Deploy
- Configuration->Permissions->Click on the Execution role name link->Add permissions->Attach policies->Search for S3->Select AmazonS3ReadOnlyAccess->Add permission
- Go to Lambda->Triggers->Add Trigger->Select a source: S3->Bucket:Select your bucket->Leave everything else as default->Acknowledge->Add
- In S3, Click on the Bucket name->Properties->Scroll down->Event notifications will show our Lambda event

Lambda Function Code

- # Create an event notification for S3 uploads
- # Write the names of files uploaded to an S3 bucket to CloudWatch Logs
- import json
- import logging
- import boto3
- # Initialize logging
- logger = logging.getLogger()
- logger.setLevel(logging.INFO)
- def lambda_handler(event, context):
- # Log the raw event
- logger.info("Event: " + json.dumps(event))
-
- # Process each record within the event
- for record in event['Records']:
- # Extract the bucket name and file key from the event
- bucket_name = record['s3']['bucket']['name']
- file_key = record['s3']['object']['key']
-
- # Log the bucket name and file key to CloudWatch
- logger.info(f"New file uploaded: {file_key} in bucket {bucket_name}")
-
- return {
- 'statusCode': 200,
- 'body': json.dumps('Processed S3 upload event successfully!')}
- }
-
- # 2. Edit the execution role to add permissions to Lambda to read from S3
- # 3. Create an event notification for all S3 object create events by adding a trigger to AWS Lambda
- # 4. Upload a file and check if a message is written to CloudWatch Logs that includes the file name

Testing Lambda Function

- Go to S3 Bucket and upload 2-3 files
- Go to Lambda->Monitor->View CloudWatch logs->Scroll down to Log streams->Click on the URL
- We should see the logs for the uploaded files

Cleanup

- **Remove all objects from the S3 bucket**
- **Remove S3 bucket**
- **Remove Lambda function**

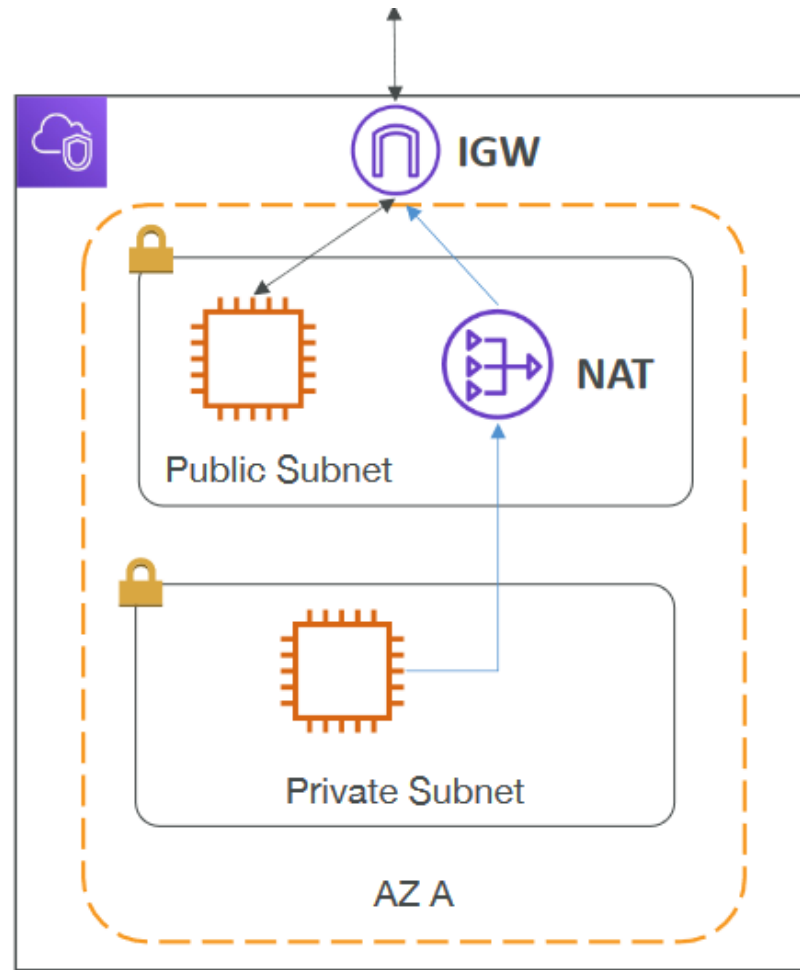
VPC

VPC

- **VPC = Virtual Private Cloud** ... Logically isolated portion of a network within an AWS region ... Like our own private network within a region
- **Subnets:** Partition our network inside a VPC
 - **Public subnet:** Accessible from the internet
 - **Private subnet:** Not accessible from the internet
- To define access to the internet and between subnets, we use **Router tables**
- **Internet Gateway:** Connect public subnet to the Internet
- **NAT Gateway:** Connect private subnet to public subnet

Internet Gateway and NAT Gateway

- **Internet gateway:** Connect the public subnet in the VPC to the Internet
- **NAT gateway (AWS-managed):** Allow private subnet in the VPC to access the Internet via the Internet Gateway



Public Subnet Route Table

Destination	Target
172.31.0.0/16	Local
0.0.0.0/0	IGW-id

Private Subnet Route Table

Destination	Target
172.31.0.0/16	Local
0.0.0.0/0	NAT-GW-id

Problem

- Create a new VPC by the name ditiss-lab with public & private subnet.
- Assign network address as 172.20.0.0/16.
- Assign 172.20.0.0/16 to the public and 172.20.10.0/24 to the private subnet.
- Create one instance and connect to a private subnet.
- Create one instance and connect to the public subnet.
- Install httpd on the private instance and check with curl from the public connected instance.

Solution

- Create a new VPC
- Type: VPC and more
- Name tag: my-ditiss-lab
- IP4 CIDR block: 172.20.0.0/16
- Number of availability zones: 1
- Number of public subnets: 1
- Number of private subnets: 1
- Open Customize subnets CIDR blocks
 - Public subnet CIDR block: 172.20.0.0/17
 - Private subnet CIDR block: 172.20.128.0/17
- NAT gateway: In 1 AZ
- Create VPC

Solution

- Create a new EC2 instance (Public)
- Name: My-EC2-**public**-instance->Key pair: Use a previous key pair->Network Settings>Edit->VPC:my-ditiss-lab->Subnet:Select the **Public** subnet->Auto-assign public IP->Enable->Launch instance
- Create a new EC2 instance (Private)
- Name: My-EC2-**private**-instance->Key pair: Use a previous key pair->Network Settings>Edit->VPC:my-ditiss-lab->Subnet:Select the **Private** subnet->Auto-assign public IP->Enable->Launch instance

Connecting to the Public Instance

- Use Puttygen to convert .ppk file into .pem on the Windows machine (Puttygen->Load->Select .ppk->Conversions->Export OpenSSH key->Save.pem file in the same directory)
- Now copy the .pem file into a Linux virtual machine using WinSCP
- On that Linux terminal: `ssh -i <.pem file> ec2-user@<public IP of public instance>`
- If unsuccessful, on the Linux terminal: `chmod 400 <.pem file name>`
- Retry above SSH
- If successful, now use WinSCP to connect to the public instance
 - Click on Advanced->SSH-Authentication->Private key file->Select .ppk file
- If successful, copy the .pem file from the Windows machine into public instance
- On the public instance: `chmod 400 <.pem file name>`
- On the public instance: `ssh -i <.pem file> ec2-user@<private IP of private instance>`

On the Private Instance

- Install Apache: **sudo yum install httpd -y**
- Start Apache service: **sudo service httpd start**
- Start this on boot: **sudo chkconfig httpd on**

- Open another putty session, SSH into the public instance and type **curl <private IP of private instance>**
- We should see Apache home page
- Public instance is called **bastion host**

Solution

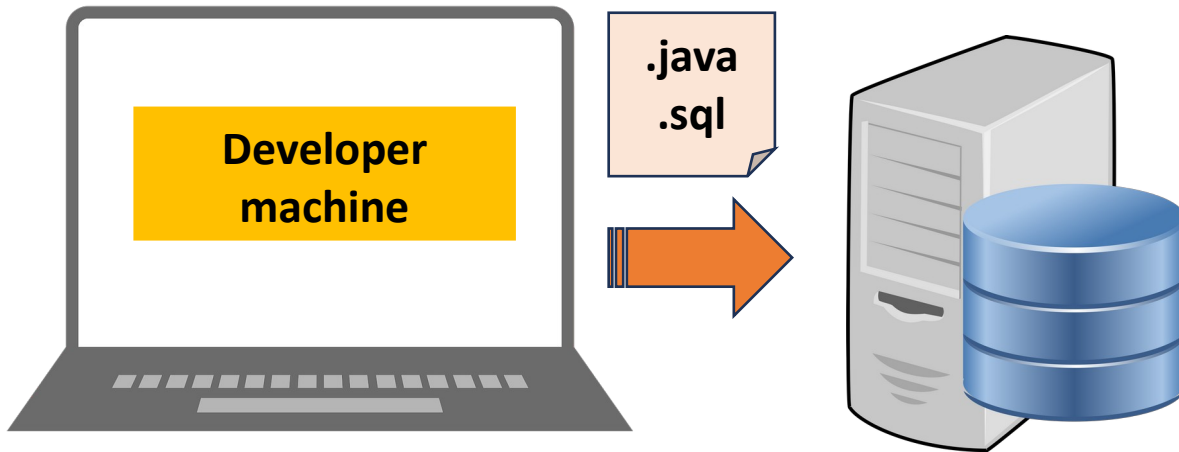
- **CLEAN UP EVERYTHING!**
- **EC2 instances**
- **Subnets**
- **Network interfaces**
- **Internet Gateways**
- **Elastic IP**

Docker: Containerisation

Atul Kahate

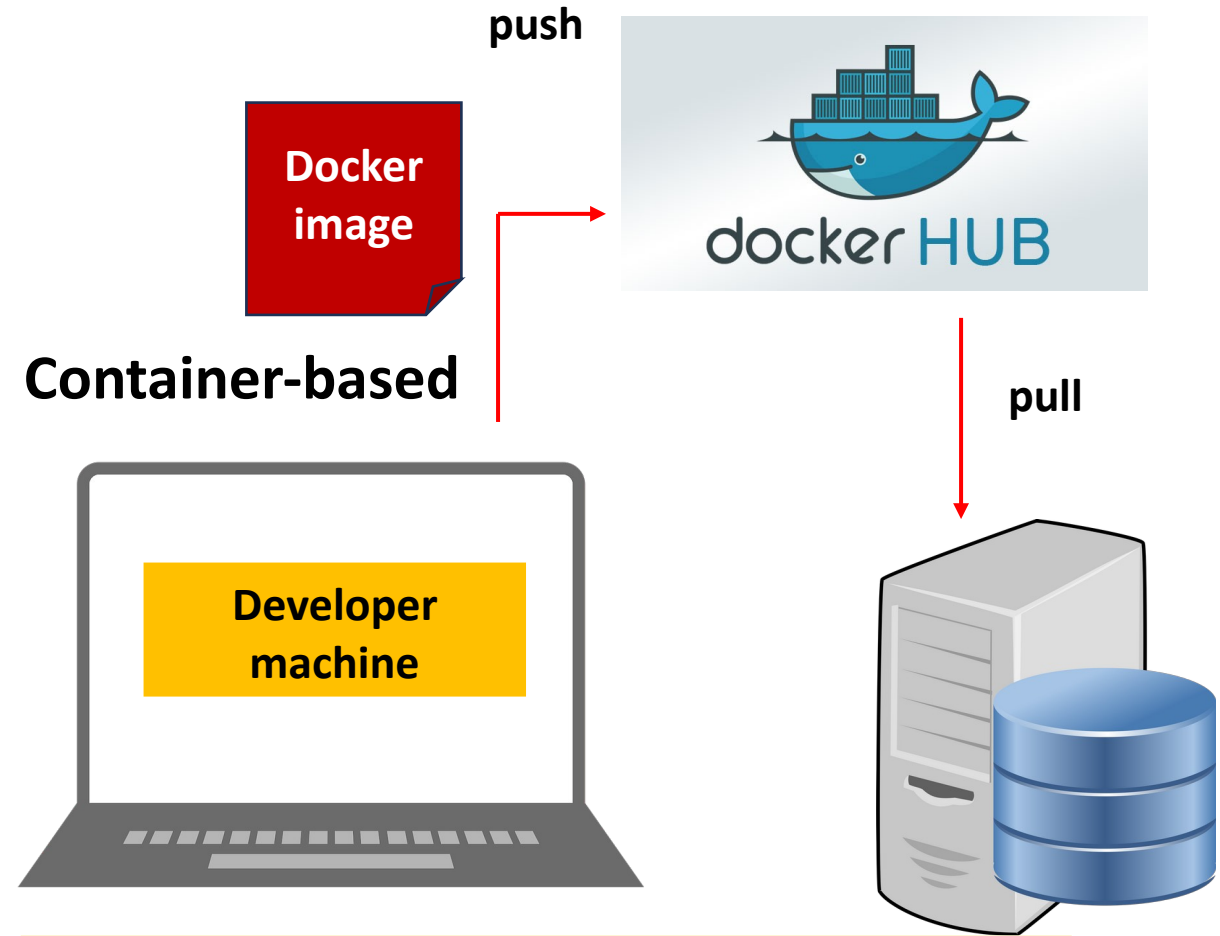
Software Deployment

Traditional



Client: Write code (.java, .sql)
Client: Compile, test on the client
Server: Install Java, MySQL, Spring Boot, ...
Client: Copy .class/.jar/.war files and .sql files on the server
Server: Test

Container-based



Client: Create microservices (.java, .sql)
Client: Compile, test on the client
Client: Create .jar/.war file and push Docker image
Server: Pull Docker image
Server: Run Docker image as a Container

What is Docker?

- **Docker:** **Container** technology
- Container = Complete and isolated software
- Runs an application in a sandbox
- Portable
- Contains all dependencies
- We can run several containers on a machine at the same time
- Self-contained: Do not alter the host system they run on
- Docker is light-weight, unlike VM
- If something works in Development, it will work in Production, too

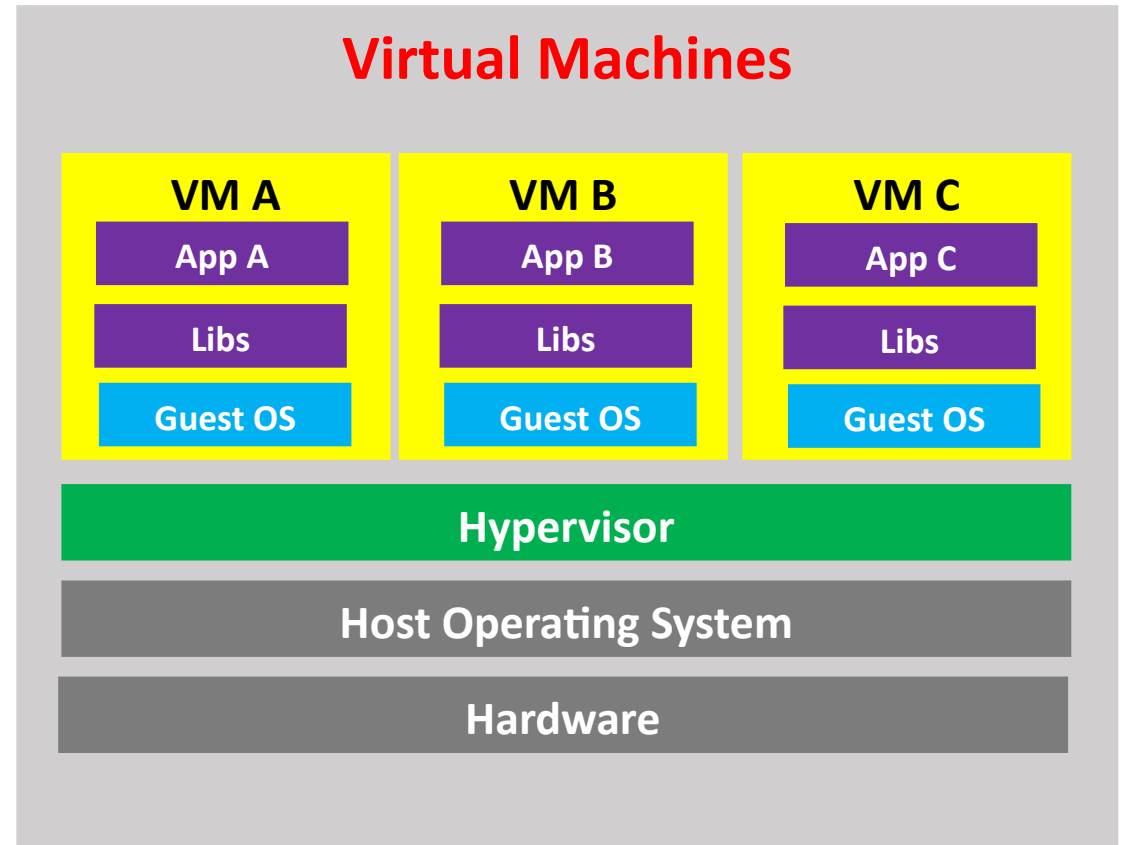
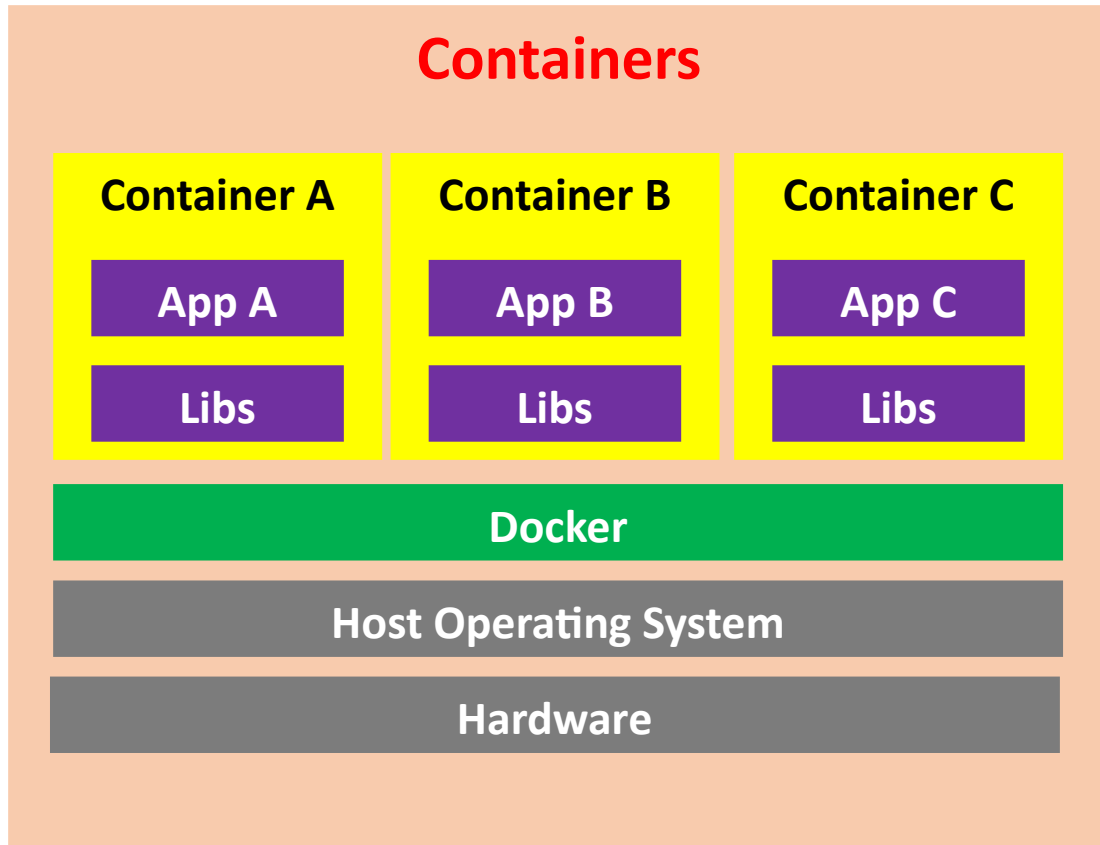


Image and Container

- **Build:** Create an **image**
- **Run:** Run the image as a **container**



Container versus Virtual Machine



Docker Installation on Ubuntu 22.04/Above

- How to find what is our current Linux distribution? **cat /etc/os-release**
- How to find what is our current Linux release? **uname -srm**
- Link:
<https://docs.docker.com/engine/install/ubuntu/#install-using-the-repository>
- Checking running status: **sudo systemctl status docker**
- Enabling Docker: **sudo systemctl enable docker**
- Starting Docker: **sudo systemctl start docker**

Solve Docker Permission Errors (Linux)

- **sudo usermod -aG docker \$USER**
- **newgrp docker**
- **groups \$USER**
- <Restart the machine>
- **docker ps**

Run a Container

- **docker run -d -it <container name>**
- **docker run ubuntu** ... Container will start and immediately stop
- **docker ps** ... See the running containers
- **docker run -it ubuntu** ... Container will open an **interactive terminal** and will run till we keep it open ... When we type exit, it will stop
- **docker run -dit ubuntu** ... Container will run in the **detached** mode (background) ... Even if we exit from **interactive terminal** (/bin/bash), it will not stop
 - How to open the interactive terminal? **docker exec -it <container id> /bin/bash**

Docker Request Limits

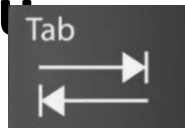
- To know our Docker request limits (Generally 100 requests in every 21600 seconds, i.e. 6 hours)
- `sudo apt install jq`
- `TOKEN=$(curl "https://auth.docker.io/token?service=registry.docker.io&scope=repository:ratelimitpreview/test:pull" | jq -r .token)`
- `cat $TOKEN`
- Whatever value we see, paste it in <https://jwt.io/> website's box

View Running Container List

- **docker containerlist** OR
- **docker ps**
- See all the running containers
- Note that there is a randomly generated container name and id

Stop a Container

- **docker stop b3**
- Here, b31 is assumed to be the ubuntu container id's first three characters
- We can also stop a container by its name
- Again run the container and stop by its name
- **docker run -dit ubuntu**
- **docker stop beautiful_joe**
- Assuming that the container name was *beautiful_joe*: Verify it stopped
- **docker container list**



View Image List

- **docker images**
- Shows list of Docker images on our local machine
- Note how much size it occupies (generally in MB)

Inspect a Container

- **docker inspect 89**
- 89 is assumed to be the start of a container's id

Pull an Image

- **docker pull nginx**
- Pull the image from DockerHub onto our local machine
- Check using **docker images**
- Note: Pulling an image does not run a container using it
- See how nginx image was created: **docker history nginx**

Delete an Image

- **docker pull nginx**
- **docker images**
- **docker rmi <image id>**
- It will *untag* the image (mark it as unused or *dangling*)
- **docker images**
- To remove the image completely (like garbage collection)
- **docker image prune**
- How much disk space is Docker using?
- **docker system df**

Give a Name to a Container

- Earlier command: **docker run -dit ubuntu**
- With name: **docker run -dit --name=my_ubuntu ubuntu**
- Verify: **docker container list**
- See all containers, running or not: **docker container list -a**
- Run with one more name:
- **docker run -dit --name=another_ubuntu ubuntu**
- See only the latest container: **docker ps -l**

Remove a Container

- First stop the container: **docker stop another_ubuntu**
- Remove: **docker rm another_ubuntu**

View Logs

- Suppose we run the hello world container in the background ... How can we see its output?
- **docker run -dit hello-world**
- **docker logs 55**
- 55: Container id

Make a Container Accessible in a Browser

- **`docker run --name my_nginx -d -p 8080:80 nginx`**
- Map the container's internal port 80 to the external (i.e. our machine's) port 8080
- It means that whatever requests we sent on our local machine's port 8080 will be sent to the container on its port 80
- Open the browser: localhost:8080
- **`docker logs my_nginx`**
- **`docker stop my_nginx`**

Custom Content in Containers: Volumes

- **mkdir webpages**
- **cd webpages**
- **echo 'Hi from inside our CDAC nginx container!' > index.html**
- **cd ..**

Volumes syntax: **Host path:Container path**

%cd% = Current directory in Windows
Same as **\${PWD}** in Linux

- **Windows:** **docker run -p 8080:80 --name=another_nginx -v %cd%/webpages:/usr/share/nginx/html:ro -d nginx**
- **Linux:** **docker run -p 8080:80 --name=another_nginx -v \${PWD}/webpages:/usr/share/nginx/html:ro -d nginx**
- **Browser:** localhost:8080

:ro means the container cannot modify the files, it can *read only*

Docker Volume

- Containers are **ephemeral/transient/disposable/stateless**
- Meaning: Short-lived, Can be easily created, destroyed, recreated
- When a container is removed, all information inside it is also destroyed
- Consider MYSQL container – Remove it and all data will also be removed
- Solution: Docker **volume** to persist data on the local disk
- When a container is removed and restarted, it will regain data from the volume

Enter and Connect to a Container

- **docker run -it --name apache httpd /bin/bash**
- Run Apache web server in the foreground
- Now we can run commands such as `pwd`, `ls`, etc inside the container
- To come out: **exit**
- **docker ps**
- The container is stopped, because we did not use the `-d` option
- **docker run -dit --name second_apache httpd /bin/bash**
- **docker ps**

Build Image Using Dockerfile

- **Dockerfile:** Contains instructions that we want to give to Docker for building an image ... Main instructions ...

Instruction	Use
FROM	Set the base image
CMD	Command to be executed when the image is run as a container (Or pass options to ENTRYPOINT)
RUN	Executes a command to help build our image
EXPOSE	Opens network ports
VOLUME	Sets a disk share
COPY	Copies files from the local disk to the image
ENV	Add an environment variable
ENTRYPOINT	Determines which executable runs when a container starts (Use CMD to pass options to the executable)

'Hello World' in Java using Docker

- `sudo mkdir ~/docker_java_example`
- `cd ~/docker_java_example`
- `sudo nano HelloWorld.java`

- `// HelloWorld.java`

```
public class HelloWorld {  
    public static void main(String[] args) {  
        System.out.println("Hello, World");  
    }  
}
```


Create Dockerfile

- **sudo nano Dockerfile**
- FROM openjdk:12-alpine
- COPY . /my_java_app
- WORKDIR /my_java_app
- RUN javac *.java
- CMD ["java", "HelloWorld"]

Build and Execute

- # Build the Docker image
- **docker build -t my-ditiss-java .**
- # Run the Docker container
- **docker run my-ditiss-java**

Pushing Docker Image to DockerHub

- **Create a user account on DockerHub**
- # On the local terminal: Login to Docker Hub
- **docker login**
- # Tag the local image
- **docker tag my-ditiss-java newdelthis/my-ditiss-java:1.0**
- # Push the image to Docker Hub
- **docker push newdelthis/my-ditiss-java:1.0**

Replace the highlighted user name with your DockerHub user name

Saving/Importing Images To/From the Local Disk

- `docker pull ubuntu`
- **Save image locally**
- `docker image save ubuntu > ubuntu.tar`
- **Copy this file on any machine and create ubuntu image**
- `docker import ubuntu.tar ubuntu.latest`

Stop and Remove All Running Containers

- **docker stop \$(docker ps -a -q)**
- **docker rm \$(docker ps -a -q)**

Problem

- Create a C program to take input from the user and test it
- Then create an image using Dockerfile
- Test image by the running container
- Then push the image to the docker hub repository
- Then also export the image in a “cprogback.tar” file

Create and Run C Program

- `sudo mkdir ~/docker_c_example`
- `cd ~/docker_c_example`
- `sudo nano user_input.c`

- `#include <stdio.h>`
- `int main() {`
- `char name[100];`
- `printf ("Enter your name: ");`
- `scanf ("%s", name);`
- `printf ("Hello, %s!\n", name);`
- `return 0;`
- `}`

Dockerfile

- **sudo nano Dockerfile**
- FROM gcc:latest
- WORKDIR /cdac/ditiss/mycode
- COPY user_input.c /cdac/ditiss/mycode
- RUN gcc -o user_input user_input.c
- CMD ["/user_input"]

Build and Execute

- # Build the Docker image

- **docker build -t my-ditiss-c .**

- # Run the Docker container

- **docker run my-ditiss-c**

PROBLEM!

- **docker run -it my-ditiss-c**

Web Application: Running a Tomcat Container

- **`docker run -dit --rm --name tomcat-container -p 8080:8080 tomcat:8.0`**
- Browser: Localhost:8080
- **`docker exec -it tomcat-container /bin/bash`**
 - `cd webapps/examples`
 - `echo "Hello from DITIIS" > cdac.html`
- Browser: localhost:8080/examples/cdac.html

Problem

- Create an httpd container running on port 8000
- Name the container as web5
- Do not map any directory
- Create an “index.html” file & copy this file inside the container and check if the website displays your webpage
- Again, modify the “index.html” file and copy it inside the container, and check if the website is updated or not

Solution

- `docker run -dit --name web5 -p 8000:80 httpd`
- On the local machine, create index.html
- `<html>`
- `<body>`
- `<h1>Hello from CDAC!</h1>`
- `</body>`
- `</html>`
- `docker cp index.html web5:/usr/local/apache2/htdocs/index.html`
- Open browser: localhost:8080
- Modify index.html to have `<h1>Hello from DITISS!</h1>`
- `docker cp index.html web5:/usr/local/apache2/htdocs/index.html`
- Open browser: localhost:8080

Docker Network

- Default: All containers run in a network called **bridge network**
- They can communicate with each other using their IP addresses
- Risky, because any container can access any other container
- Solution: Create our own **custom network** and add our containers to it
- Advantage: Now our containers are isolated from the others and can access each other by their names also (not just IP addresses)

Docker Volume and Docker Network Example

- Run a Tomcat container and a MySQL container
- Put them in the same Docker network
- From the Tomcat container, create a *Student* table in MySQL container
- It will have three columns: PRN, Name, Course



Docker Networking Example

- **docker network create my-network**
- **docker container run --name mysql-container -v /data/mysql_data:/var/lib/mysql --network my-network -e MYSQL_ROOT_PASSWORD=pass -d mysql:latest**
- **docker container run --name tomcat-container --network my-network -p 7777:8080 -v /data/mytomcat_data:/usr/local/tomcat/webapps -d tomcat:8**

Docker Networking Example

- Connect to the Tomcat container and verify that it can access MySQL
 - **`docker exec -it tomcat-container /bin/bash`**
 - **`apt-get update`**
 - **`apt-get install -y mysql-client`**
 - **`mysql -h mysql-container -uroot -ppass`**
- Create a database, a table, and add 1 row:
 - `CREATE DATABASE ditiss;`
 - `USE ditiss;`
 - `CREATE TABLE student_tbl (prn integer, name char(30));`
 - `INSERT INTO student_tbl VALUES (1, "My name");`
 - `SELECT * FROM student_tbl;`

Docker Compose

- **Docker compose:** Tool for automating the building and running of containers
- How: Create a docker-compose.yml file
- docker-compose.yml: Replaces the **docker build** and **docker run** commands
- Dockerfile: Will still be needed for the actual step-by-step instructions

Dockerfile

Usual details for creating image

docker-compose.yml

Replaces the **docker build** and **docker run** commands

docker compose build
docker compose up
docker compose down



HelloWorld.java

- **sudo nano HelloWorld.java**
- public class HelloWorld {
- public static void main(String[] args) {
- System.out.println("Hello, World!");
- }
- }

Dockerfile

- **sudo nano Dockerfile**
- FROM openjdk:17-alpine
- WORKDIR /my-compose-app
- COPY HelloWorld.java /my-compose-app
- RUN javac HelloWorld.java
- CMD ["java", "HelloWorld"]

docker-compose.yml

- **sudo nano docker-compose.yml**
- services:
- java-helloworld:
- build: .
- container_name: java-helloworld-container
- command: ["java", "HelloWorld"]

Test

- **docker compose up**
- **docker compose down**

Docker Swarm

- **Docker Swarm:** Docker's built-in **container orchestration** tool
- Alternative to Kubernetes
- Not so popular

- **docker info**
- If it shows Docker Swarm not active: **docker swarm init**
- Swarm: Single-node cluster
- **docker node ls**

Docker Swarm Service Example

- `docker service create nginx`
- Get more details: `docker service ls`
- `docker service ps <service id>`
- `docker service update <service id> --replicas <Number>`
- `docker ps`
- `docker container stop <container id>`
- `docker container rm -f <container id>`
- `docker container ls` ... Note that a replacement container has started
- `docker service rm <service name>` ... Remove the service itself

service: We create a service in Docker Swarm to run containers inside it ... It is a logical group of multiple copies of the same container

Docker Swarm Problem

- Create a new docker image using NGINX
- Push it to docker hub
- Create a service on Dockerswarm
- Keep initial replicas as 1 and then increase the replicas to 10
- Reduce the replicas to 2
- Create a new image by modifying “index.html”
- Push to docker hub and then update the service for a new image

Solution – 1

- `mkdir nginx_project`
- `cd nginx_project`
- `echo "<h1>Welcome to my custom NGINX server</h1>" > index.html`
- `sudo nano Dockerfile`
- `FROM nginx:latest`
- `COPY index.html /usr/share/nginx/html/index.html`
- `docker build -t newdelthis/custom-nginx:1.0 .`
- `docker run -d -p 9999:80 newdelthis/custom-nginx:1.0`
- Browser: localhost:9999
- `docker login`
- `docker push newdlethis/custom-nginx:1.0`

Solution – 2

- `docker swarm init`
- `docker service create --name nginx-service --replicas 1 -p 9999:80 newdelthis/custom-nginx:1.0`
- `docker service ls`
- `docker service scale nginx-service=10`
- `docker service ps nginx-service`
- `docker service scale nginx-service=2`
- `docker service ps nginx-service`

Solution – 3 (New index.html)

- `echo "<h1>Welcome to the updated NGINX server</h1>" > index.html`
- `docker build -t newdelthis/custom-nginx:2.0 .`
- `docker push newdelthis/custom-nginx:2.0`
- `docker service update --image newdelthis/custom-nginx:2.0 nginx-service`
- `docker service ps nginx-service`
- Browser: localhost:9999
- `docker service rm nginx-service`

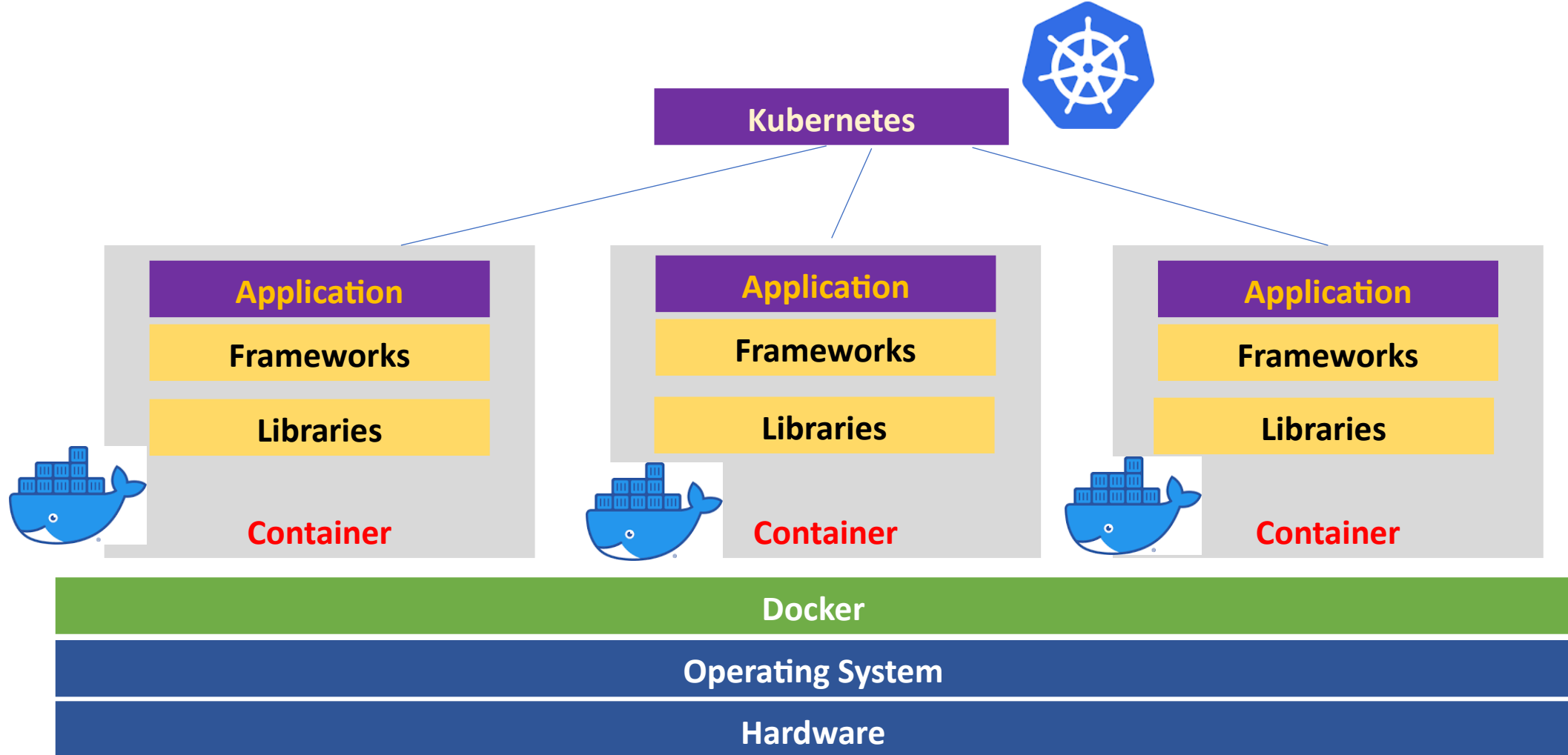
Running a Container and Stopping it After Displaying a Message

- Run an Alpine container that displays *Hello World*
- `docker run --rm alpine sh -c "echo 'hello world'"`

Kubernetes (K8S)

Atul Kahate

Kubernetes and Container Orchestration

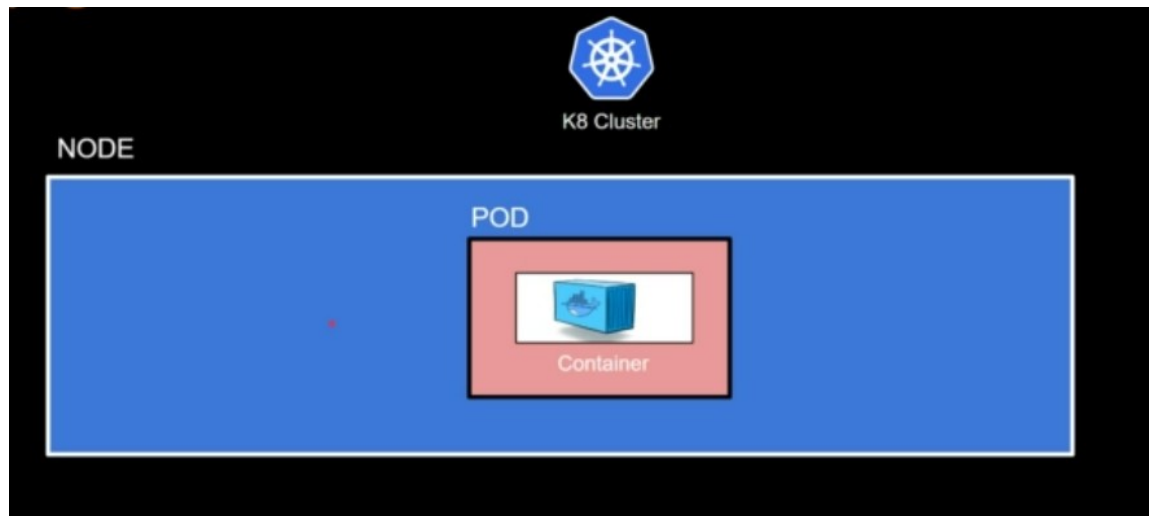


Why Container Orchestration?

- **Requirement:** We want 10 instances of **Microservice A** container, 15 instances of **Microservice B** container and
- **Typical Features:**
 - **Auto Scaling** - Scale containers based on demand
 - **Service Discovery** - Help microservices find one another
 - **Load Balancer** - Distribute load among multiple instances of a microservice
 - **Self Healing** - Do health checks and replace failing instances
 - **Zero Downtime Deployments** - Release new versions without downtime

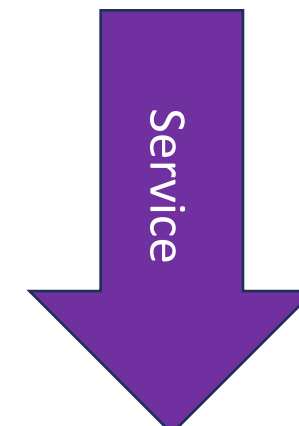
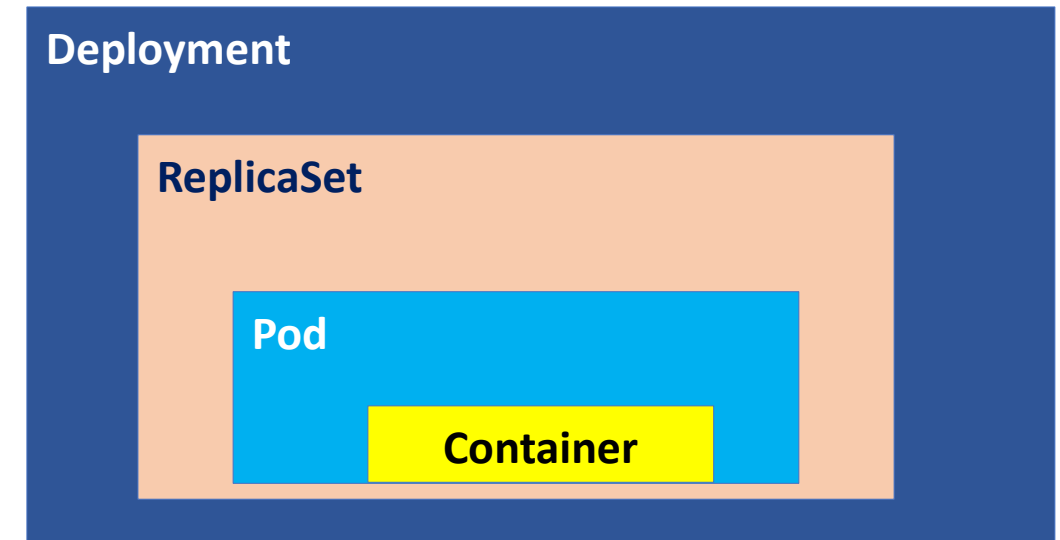
Kubernetes Architecture

- Physical architecture

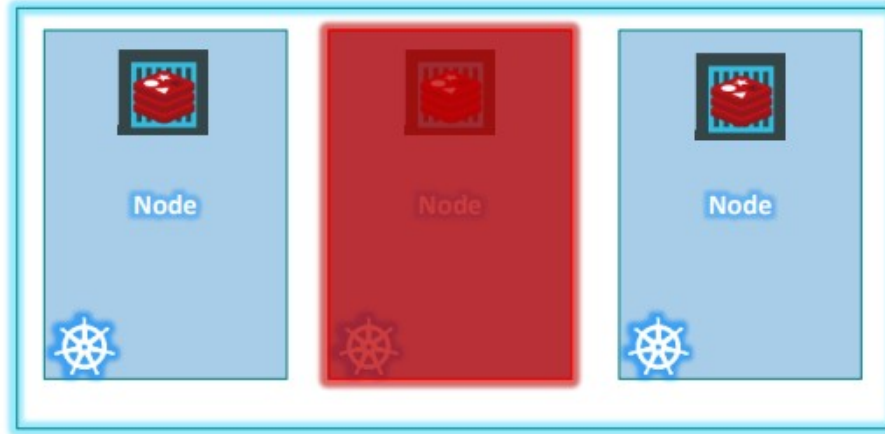


- Remember **NPC** – **N**ode contains **P**od contains **C**ontainer

- Logical architecture



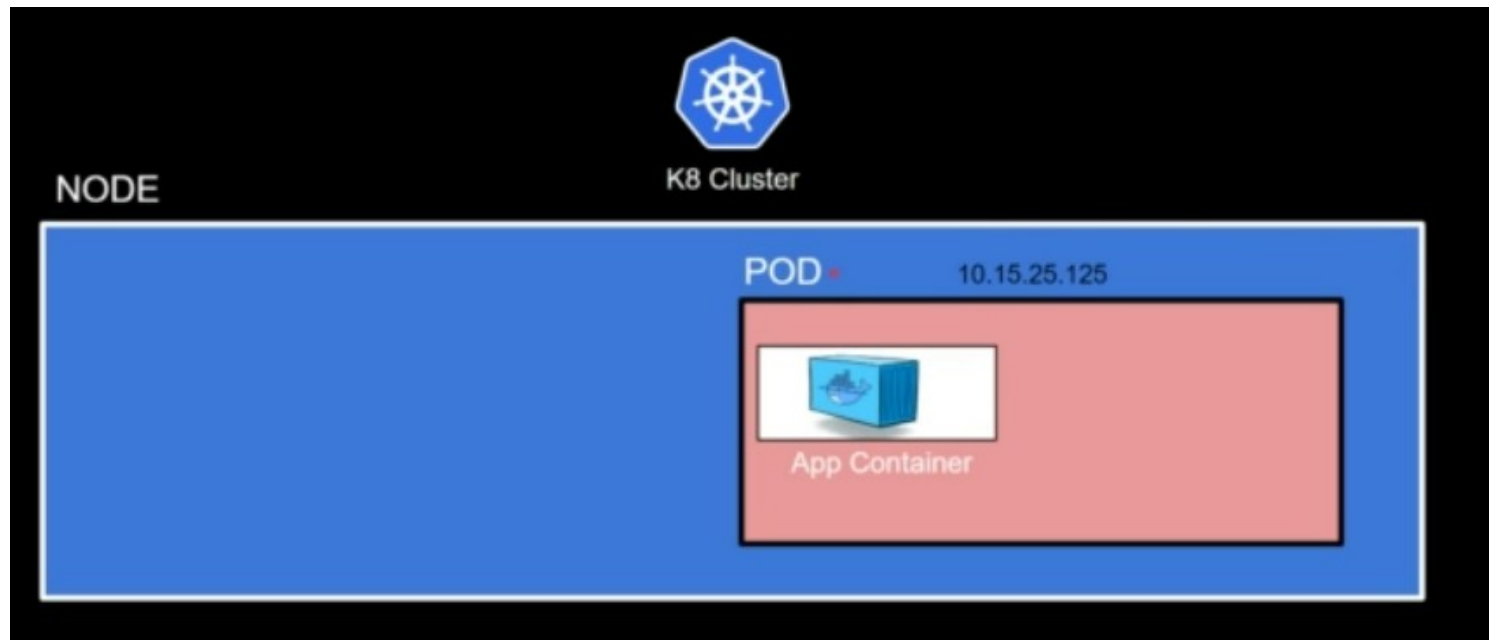
Physical Architecture: Cluster. Nodes



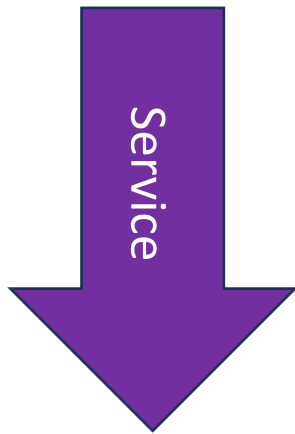
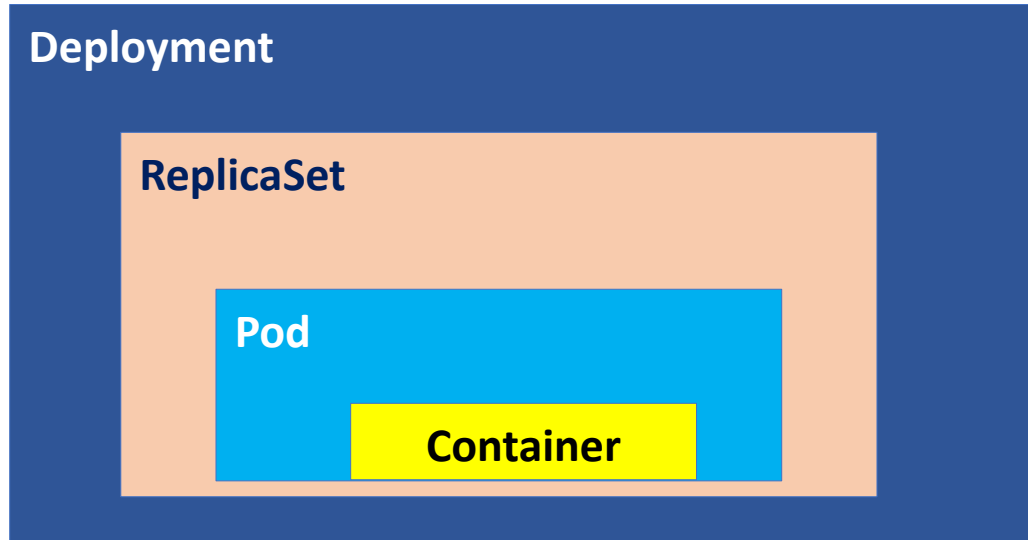
- **Cluster**: Gets created when we install Kubernetes
- Contains One/more **master nodes** and one or more **worker nodes**
- **Master node**: Performs management
- **Worker node**: Contains **pods**
- If one node fails, the other takes over
- Also helps in **load balancing**

Physical Architecture: Pod

- **Pod:** A Single instance of an application
- Docker = Image->Container, Kubernetes = Container->Pod
- Pod = Container with an internal/a private IP address



Logical Architecture



Object

Use

Pod

A Docker container with an IP address

Replicaset

A group of identical pods

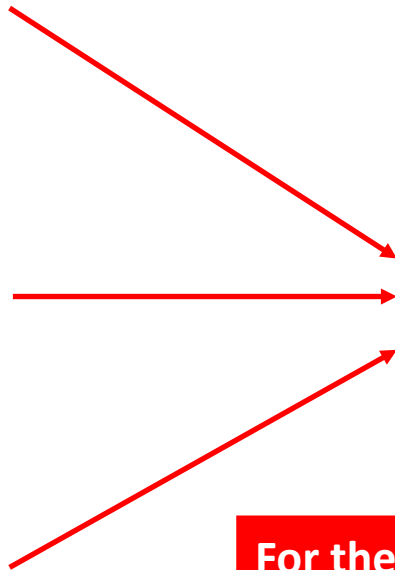
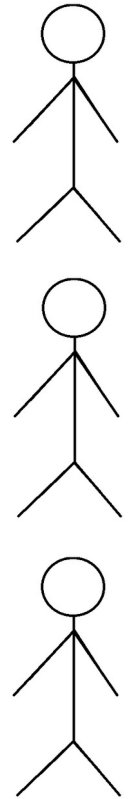
Service

Exposes a replicaset to the outside world with a consistent external IP address

Deployment

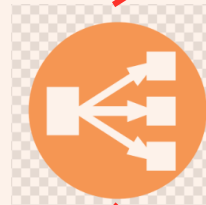
Overall management structure

Workflow

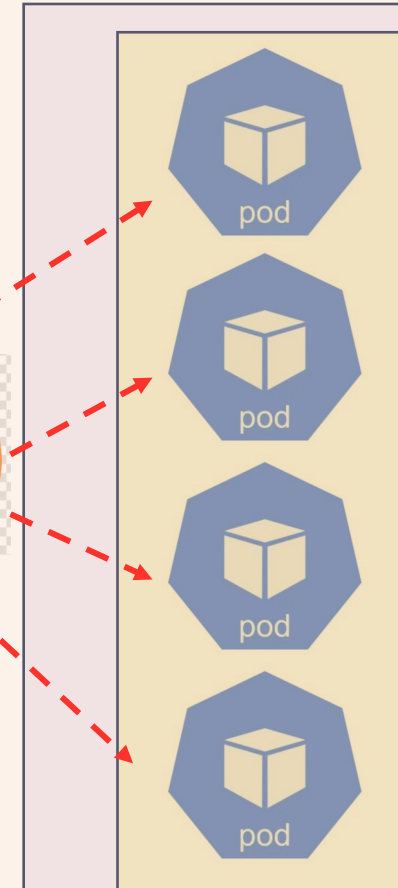


**Static IP
address**

**For the entire application,
there will be one service, so
one static IP address**



**E-Commerce
Application**



**One Deployment = One
Microservice**

**So, we will have many
Deployments in this application**

Kubernetes Installation

- Cluster options
 - Windows: Enable **Kubernetes** in **Docker Desktop**
 - Linux: Use **Microk8s**, since Minikube gives problems (<https://microk8s.io/>)
- Command line
 - The **kubectl** utility (<https://kubernetes.io/docs/tasks/tools/>)

Linux Installation (Ubuntu 22.04 LTS) using Microk8s (A Kubernetes Engine)

- **sudo apt install snapd**
- **sudo systemctl enable --now snapd**
- **sudo snap install microk8s --classic --channel=1.28**
- **sudo usermod -a -G microk8s \$USER**
- **sudo chown -f -R \$USER ~/.kube**
- **su - \$USER**
- **microk8s status --wait-ready**
- **microk8s kubectl get nodes**
- **microk8s kubectl get services**
- **alias kubectl='microk8s kubectl'**
- **kubectl get nodes**

kubectl command

kubectl

[command]

create
get
describe
delete
apply
attach
autoscale

And Many
More...

[type]

Any K8 Resource Type
pods OR pod OR po
namespaces OR ns
deployments OR deploy
replicasets OR rs

And Many More....

[name]

Optional
Name of the resource.
If omitted, all resource
details displayed

[flags]

Optional
--filename OR -f
--output OR -o

And Many More...

Using Kubernetes

Create a Pod Directly or Using a Deployment

- **Run a pod**
- `kubectl run my-nginx --image nginx` ... We must give a name to every pod
- `kubectl get pods`
- **Create a deployment (and a pod inside it)**
- `kubectl create deployment my-nginx --image nginx`
- `kubectl get pods` ... Note how the two nginx pods look different
- `kubectl get all` ... Note a replicaset was automatically created by the deployment
- **Cleanup**
- `kubectl delete pod my-nginx`
- `kubectl delete deployment my-nginx`

Replicaset and Service

- **kubectl create deployment my-apache --image httpd**
- **kubectl scale deployment my-apache --replicas 2**
- **kubectl expose deployment my-apache --type=NodePort --port=80**

Main Service Types

Service Type	Details	Use case
ClusterIP	Default ... Pods are accessible only inside the cluster ... Service gets an IP address and all the pods also get the same IP address ... When the service receives requests, it load balances them among the pods ...	Backend pod (e.g. MySQL) can have a ClusterIP, meaning it can only be accessed from the inside by the frontend web application ... Cannot be accessed by anyone from outside ...
NodePort	Accessible from outside the cluster ... A static port (NodePort) on the Kubernetes cluster is used to expose the service ... No load balancing takes place ...	A developer may want to test the application before exposing it using LoadBalancer
LoadBalancer	Accessible from the outside with an external static IP address ...	A real application hosted on a public server/cloud

Deployment using YAML Files

- Declarative approach
- Mostly used in the real world
- Command: **kubectl apply -f <filename.yml>**
- Each file contains one or more **manifests**
- **Manifest:** Describes one API object (deployment, job)
- Each manifest has four parts
 - **apiVersion:**
 - **kind:**
 - **metadata:**
 - **spec:**

Sample YAML file (C:\lectures\CDAC\DAC\nginx-pod-demo.yml)

- apiVersion: v1
- kind: Pod
- metadata:
 - name: nginx
- spec:
 - containers:
 - - name: nginx
 - image: nginx:1.17.3

This YAML file can be used to deploy nginx into a pod

kubectl apply -f nginx-pod-demo.yml

kubectl delete -f nginx-pod-demo.yml

Sample YAML file (C:\lectures\CDAC\DAC\nginx-deployment-demo.yml)

- apiVersion: apps/v1
- kind: Deployment
- metadata:
- name: nginx-deployment
- spec:
- selector:
- matchLabels:
- app: nginx
- replicas: 2
- template:
- metadata:
- labels:
- app: nginx
- spec:
- containers:
- - name: nginx
- image: nginx:1.17.3
- ports:
- - containerPort: 80

This YAML file can be used to deploy nginx as a deployment

kubectl apply -f nginx-deployment-demo.yml

kubectl delete -f nginx-deployment-demo.yml

Port Forwarding

- When a pod is deployed inside a deployment but without a service, it is not accessible outside of the cluster
- Try: localhost:80 in the browser ... Not accessible
- Solution: **Port forwarding**: Forward requests coming to the pod to another port on the local host
- Example: Our nginx deployment has deployed nginx pod on port 80, so port number inside the container is 80
- Run this command: **kubectl port-forward** nginx-deployment-7b767787b6-246tl **9999:80** (Replace pod id with one of the actual pod ids by doing **kubectl get pods**)
- Now try 127.0.0.1:9999 in the browser – Should show nginx page

Sample YAML file (C:\lectures\CDAC\DAC\nginx-service-demo.yml)

```
• apiVersion: apps/v1
• kind: Deployment
• metadata:
•   name: my-nginx
•   labels:
•     app: nginx
• spec:
•   replicas: 3
•   selector:
•     matchLabels:
•       app: nginx
•   template:
•     metadata:
•       labels:
•         app: nginx
•     spec:
•       containers:
•         - name: nginx
•           image: nginx:latest
•           ports:
•             - containerPort: 80
• ---
• apiVersion: v1
• kind: Service
• metadata:
•   name: my-nginx-service
• spec:
•   type: NodePort
•   ports:
•     - port: 80
•   selector:
•     app: nginx
```

This YAML file can be used to deploy nginx
as a service

kubectl apply -f nginx-service-demo.yml

kubectl delete -f nginx-service-demo.yml

Opening a Terminal Inside a Pod

- First get a pod's id: **kubectl get pods**
- Open a terminal inside it: **kubectl exec -it my-nginx-7c79c4bf97-bzrq9 /bin/bash**
- Now we can run any Linux command
- Example: **hostname** **hostname -i** **ls**
- What application is the Pod running? **curl localhost:80**
- It should show the HTML code for default nginx home page

Scaling the Application

- Check pods first: **kubectl get pods -l app=nginx**
- Change replicas from 2 to 4 in the nginx-deployment-demo.yml file and redeploy and again check the pods
- `kubectl apply -f nginx-deployment-demo.yml`
- `kubectl get pods -l app=nginx`
- Try deleting pods manually and see what happens

Cleanup (If needed)

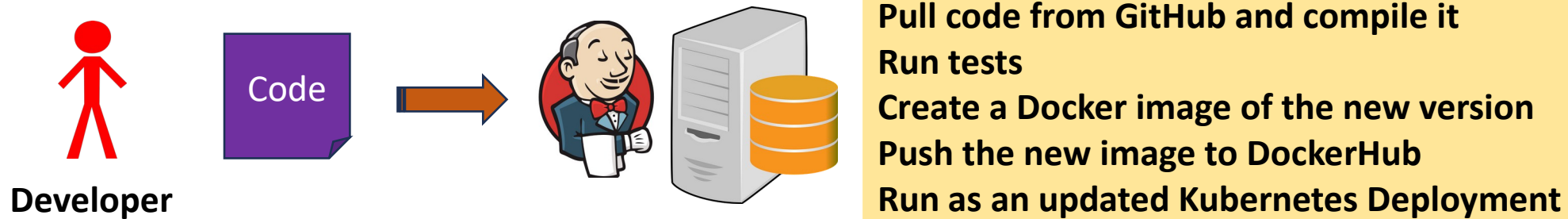
- **kubectl delete deployment <deployment name>**
- **kubectl delete service <service name>**
- To delete everything (**DANGEROUS**)
- **kubectl delete --all all -n default**

Jenkins

Atul Kahate

What is Jenkins?

- **Jenkins:** An open source **automation server**
- Used for building, deploying, and automating projects



- **Continuous Integration and Continuous Delivery (CI/CD):** Integrate + Build + Package + Deploy
- Automates code compilation, testing, deployment
- Allows for many plug-ins to enhance functionality

Jenkins Installation

- `sudo apt-get update`
- `sudo apt install fontconfig openjdk-17-jre` This will install the JRE
- `sudo apt install openjdk-17-jdk-headless` This will install JDK
- Refer to: <https://www.jenkins.io/doc/book/installing/linux/#debianubuntu> and execute all the steps ...
Then run the following ...
- `sudo systemctl enable jenkins`
- `sudo systemctl start jenkins`
- `sudo systemctl status jenkins`
- Browser: localhost:8080
- Paste password from `/var/lib/jenkins/secrets/initialAdminPassword`
- Select Install suggested plugins
- Create user: Just fill admin everywhere, give any dummy email id
- Instance configuration: Do not change

“Hello Jenkins”

- Create a new job in Jenkins
- Name: hello_jenkins
- Freestyle project
- Add Build step -> Execute shell ... In the text area, type *echo “Hello Jenkins!”*
- Save
- Left side menu: Build Now
- Scroll down to see Build History -> #1 -> Click -> Console output

Modifying the *Hello* Example

- Modify the *Execute shell* to have this:
 - NAME="Atul"
 - echo "Hello, \$NAME. Current date and time is \$(date)."
 - Rebuild the job and check output
-
- Modify the *Execute shell* to have this:
 - NAME="Atul"
 - echo "Hello, \$NAME. Current date and time is \$(date)." > /tmp/info
 - Rebuild the job and check output
 - Where is the file?
 - On the terminal: **cat /tmp/info**

Pipeline Project

- **Pipeline:** Stages of processing, can be declared in Jenkinsfile
- Dashboard->+New Item->Enter an item name: hello_Jenkins_pipeline->Pipeline->Pipeline: Definition: Pipeline script->Script: Try sample pipeline...->Hello World->Save
- Build Now->History->Console Output
- Then open the job configuration, add a second line:
- `echo 'Hello World'`
- `sh 'echo "Hello from C-DAC"'`
- `sh 'whoami'`
- Build Now->History->Console Output

Another Pipeline

- Modify the pipeline to have this:
- **echo 'Assemble a new laptop ...'**
- **sh 'mkdir -p assemble'**
- **sh 'touch assemble/computer.txt'**
- **sh 'echo "Motherboard" >> assemble/computer.txt'**
- **sh 'cat assemble/computer.txt'**
- **sh 'echo "Display" >> assemble/computer.txt'**
- **sh 'cat assemble/computer.txt'**

Workspace

- Where was the *computers.txt* file created?
- Open the output of a job
- Workspaces-><http://localhost:8080/job/test-pipeline/5/execution/node/3/ws/>->assemble->computers.txt
- Now run the job multiple times and observe the contents of computers.txt file at the end ... We will see accumulation of the contents
- How to clean up/remove the file after every run?
- After stages {} block, add a new block:
- post {
- always {
- cleanWs()
- }
- }
- Rerun the job and check ... File should be gone ... But what if we want to retain it?

Storing Built Artifacts in Jenkins

```
• pipeline {  
•   agent any  
  
•   stages {  
•     stage('Hello') {  
•       steps {  
•         cleanWs()  
•         echo 'Assemble a new laptop ...'  
•         sh 'mkdir -p assemble'  
•         sh 'touch assemble/computer.txt'  
•         sh 'echo "Motherboard" >> assemble/computer.txt'  
•         sh 'cat assemble/computer.txt'  
•       }  
•     }  
•   }  
  
•   post {  
•     success {  
•       archiveArtifacts artifacts: 'assemble/**'  
•     }  
•   }  
• }
```

After running the job,
check if the artifacts are
archived or not

Syntax for Combining Multiple Shell Commands

- sh '''
- mkdir -p assemble
- touch assemble/computer.txt
- echo "Motherboard" >> assemble/computer.txt
- cat assemble/computer.txt
- '''

Pipeline Stages: Add a Test Stage using Linux Test Command (Checks if a file is present)

```
• pipeline {  
•   agent any  
  
•   stages {  
•     stage('Create') {  
•       steps {  
•         cleanWs()  
•         echo 'Assemble a new laptop ...'  
•         sh '''  
•           mkdir -p assemble  
•           touch assemble/computer.txt  
•           echo "Motherboard" >> assemble/computer.txt  
•           cat assemble/computer.txt  
•           echo "Display" >> assemble/computer.txt  
•           cat assemble/computer.txt  
•         '''  
•       }  
•     }  
•     stage('Test') {  
•       steps {  
•         echo 'Testing the new laptop ...'  
•         sh 'test -f assemble/computer.txt'  
•       }  
•     }  
•   }  
•   post {  
•     success {  
•       archiveArtifacts artifacts: 'assemble/**'  
•     }  
•   }  
• }
```

Jenkins-Git and GitHub

- `cd ~`
- Go to GitHub and create a new repository
- `git clone <Repository URL>`
- The repository will be empty
- `cd <directory of the repository>`
- `sudo nano say_hello.sh`
- `#!/bin/bash`
- `echo Hello Jenkins from GitHub!`
- Save and exit
- `sudo chmod u+x say_hello.sh`

Write Code and Push to GitHub

- `git config --global user.name "Atul Kahate"`
- `git config --global user.email newdelthis@gmail.com`
- `git add say_hello.sh`
- `git commit -m "First commit of say hello"`
- `git log`
- Go to GitHub.com->Settings->DeveloperSettings->Personal access tokens->Tokens(classic)->Generate new token->Generate new token(classic)->Select No expiration->Check all checkboxes->Generate token->Copy this value and save in some file
- `git push origin main`
- Enter your GitHub id and paste the token
- Check on GitHub if the repository now contains say_hello.sh

Setting up Jenkins to use GitHub

- Jenkins UI->Manage Jenkins->Plugins->Available plugins->Search for Github->Select GitHub Integration->Install->Go back to the top page
(Note: Do NOT select Restart jenkins ...)
- Dashboard->New item
- Project name: my-github-project->Freestyle project->Ok->Source code management->Git->Repository URL->*Paste the actual URL*->Branches to build->Branch specifier->*/main->Save->Build now
- Configure the project->Build steps->Add build step->Execute shell->./say_hello.sh->Save->Build now

Poll SCM

- In the job configuration, check the box next to *Poll SCM*
- Under schedule:
* * * * *
- Every minute
- Every hour
- Every day of the month
- Every month
- Every day of the week
- Another example: every Monday, 8 am: 0 8 * * 1

Using Variables

- pipeline {
- agent any
-
- environment {
- INSTITUTE = "CDAC"
- CENTER = "ACTS"
- }
-
- stages {
- stage('Build') {
- steps {
- sh 'echo \$INSTITUTE \$CENTER'
- }
- }
- }
- }

Using Credentials

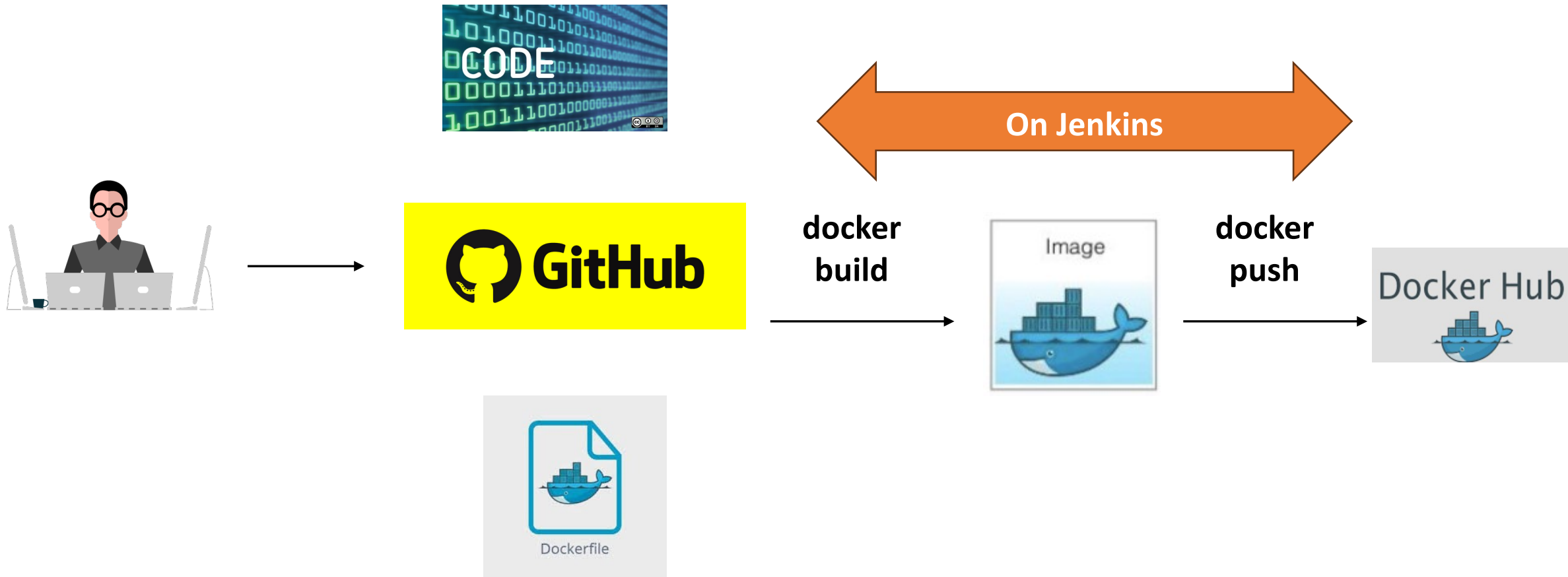
- pipeline {
- agent any
-
- environment {
- secret = credentials('SECRET_TEXT')
- }
-
- stages {
- stage('Example Stage 1') {
- steps {
- sh 'echo \$secret'
- }
- }
- }
- }

Before running this job: Dashboard->Manage Jenkins->Credentials->Dropdown after *(global)* under Domain->Add credentials->Username:cdac->Password:cdac@1234->ID:SECRET_TEXT->Save

Example: Build and Run Hello.java

- Create a new repository on GitHub named jenkins_java
- Local machine: `cd ~ ... git clone <repository id>`
- `sudo nano HelloWorld.java`
- Repeat git add, commit, push to push the file to the same repository as earlier
- Create a project in Jenkins, use GitHub settings as before
- In Build steps->Execute shell:
- **`javac HelloWorld.java`**
- **`chmod u+x HelloWorld.class`**
- **`java HelloWorld`**
- Save and build job

Jenkins and Docker



Give Permissions and Install Docker Plug-in

- By default, Docker commands can be run only by users who are a part of the docker group ... Make Jenkins a part of this group ...
- **sudo usermod -aG docker jenkins**
- Restart Jenkins:
- **sudo systemctl restart jenkins**
- Dashboard->Manage Jenkins->Plugins->Available plugins->Search for **docker pipeline**->Select and install
- Also install the **GitHub Integration** plugin

Build Docker Image: Java Hello World

- `mkdir ~/jenkins_docker_java_hello_world`
- `cd ~/jenkins_docker_java_hello_world`
- `sudo nano Jenkinsfile` ... Contents on the next slide
- `sudo nano HelloWorld.java`
- `sudo nano Dockerfile`
- On GitHub, create a repository: `jenkins_docker_java_hello_world`
- Local machine: `git init`
- `git add .`
- `git commit -m 'Jenkins with Docker'`
- `git branch -M main`
- `git remote add origin https://github.com/newdelthis/ jenkins_docker_java_hello_world.git`
- `git push -u origin main`

Build Docker Image: Java Hello World

- Go back to Jenkins UI->New Item
- Item name: **jenkins_docker_java_hello_world**->Select Pipeline->Ok
- Pipeline->Definition:Pipeline script from SCM (Because it is coming from GitHub)->SCM: Git->Repository URL:
https://github.com/newdelthis/jenkins_docker_java_hello_world.git->Branch Specifier->*/main->
- Build now

Jenkinsfile

```
• pipeline {
•   agent any

•
•   environment {
•     DOCKER_IMAGE = 'hello-world-java:latest' // Docker image name
•   }

•
•   stages {
•     stage('Checkout') {
•       steps {
•         checkout scm
•       }
•     }

•     stage('Build') {
•       steps {
•         sh 'javac HelloWorld.java'
•       }
•     }

•     stage('Package') {
•       steps {
•         sh 'jar cf HelloWorld.jar HelloWorld.class'
•       }
•     }

•     stage('Docker Build') {
•       steps {
•         sh """
•         docker build -t $DOCKER_IMAGE .
•         """
•       }
•     }
•   }

•   post {
•     success {
•       echo 'Build completed successfully.'
•     }
•     failure {
•       echo 'Build failed.'
•     }
•   }
• }
```

HelloWorld.java

- public class HelloWorld {
- public static void main(String[] args) {
- System.out.println("Hello, World!");
- }
- }

Dockerfile

- FROM openjdk:11-jdk-slim
- WORKDIR /app
- COPY HelloWorld.class /app/HelloWorld.class
- CMD ["java", "HelloWorld"]

Docker Build and Push Image in Jenkins

- `mkdir ~/docker_jenkins_demo`
- `cd ~/docker_jenkins_demo`
- `sudo nano app.py`

```
from flask import Flask  
app = Flask(__name__)
```

```
@app.route("/")  
def hello_world():  
    return "Hello from Git+Docker+Jenkins"
```

- `sudo nano requirements.txt`
`flask`

Docker Build and Push Image in Jenkins

- **sudo nano Dockerfile**
- FROM python:3.9-slim-buster
- WORKDIR /ditiss_python
- COPY . /ditiss_python
- RUN pip install -r requirements.txt
- EXPOSE 5000
- CMD ["python", "app.py"]

Docker Build and Push Image in Jenkins

```
• sudo nano Jenkinsfile

• pipeline {
•   agent any

•   environment {
•       GIT_REPOSITORY_URL = 'https://github.com/newdelthis/docker_jenkins_demo.git'
•       DOCKER_IMAGE_NAME = 'newdelthis/docker_jenkins_demo'
•       IMAGE_TAG = '1.0'
•   }

•   stages {
•       stage('Clone Repository') {
•           steps {
•               script {
•                   try {
•                       git branch: 'main', url: GIT_REPOSITORY_URL
•                   } catch (Exception e) {
•                       echo "Failed to clone repository: ${e.message}"
•                       error "Failed to clone repository"
•                   }
•               }
•           }
•       }

•       stage('Build Docker Image') {
•           steps {
•               script {
•                   try {
•                       docker build["${DOCKER_IMAGE_NAME}:${IMAGE_TAG}"]
•                   } catch (Exception e) {
•                       echo "Failed to build Docker image: ${e.message}"
•                       error "Failed to build Docker image"
•                   }
•               }
•           }
•       }

•       stage('Push to DockerHub') {
•           steps {
•               script {
•                   try {
•                       withCredentials([usernamePassword(credentialsId: 'my-docker-hub-credentials-id',
•                           usernameVariable: 'DOCKER_USERNAME',
•                           passwordVariable: 'DOCKER_PASSWORD')]) {
•                           // Explicit login before push
•                           sh """
•                           echo "${DOCKER_PASSWORD}" | docker login -u "${DOCKER_USERNAME}" --password-stdin
•                           docker push ${DOCKER_IMAGE_NAME}:${IMAGE_TAG}
•                           """
•                       }
•                   } catch (Exception e) {
•                       echo "Failed to push Docker image to registry: ${e.message}"
•                       error "Failed to push Docker image"
•                   }
•               }
•           }
•       }
•   }
• }
```

Push the Code onto GitHub

- Create a new repository on GitHub named `docker_jenkins_demo` under your user id
- Go back to the Jenkins Ubuntu machine terminal
- **`git init`**
- **`git add .`**
- **`git commit -m "first commit"`**
- **`git branch -M main`**
- **`git remote add origin`
`https://github.com/newdelthis/docker_jenkins_demo.git`**
- **`git push -u origin main`**

New Item in Jenkins

- Go back to Jenkins UI->New Item
- Item name: build-publish-docker-image->Select Pipeline->Ok
- Pipeline->Definition:Pipeline script from SCM (Because it is coming from GitHub)->SCM: Git->Repository URL:
https://github.com/newdelthis/docker_jenkins_demo.git->Branch Specifier->*/main->

Create a Credential for DockerHub

- Jenkins UI->Dashboard->Manage Jenkins->Credentials->Under Domain global->Add credentials->Scope: Retain Global->Username: Your DockerHub id->Password: Your DockerHub password->ID: **my-docker-hub-credentials-id**->Save
- *Note: my-docker-hub-credentials-id must match with the value in this line of Jenkinsfile: withCredentials([usernamePassword(credentialsId: **my-docker-hub-credentials-id**, usernameVariable: 'DOCKER_USERNAME', passwordVariable: 'DOCKER_PASSWORD')) {*

Test

- If everything is ok, the new job should run successfully when you say *Build now*
- It should pull the code from GitHub, build a Docker image, and push it on to DockerHub

Problem

- Create a CI/CD pipeline using Jenkins, Docker, and GitHub to automate the web by using the NGINX image.

Solution

- Create a new GitHub repository, clone on to the local machine, add index.html (**Hello from C-DAC**) and Dockerfile and push it back to GitHub
- Dockerfile:
 - **FROM nginx:latest**
 - **COPY index.html /usr/share/nginx/html/**
 - **EXPOSE 80**
- In Jenkins, ensure Docker and Git plugins are installed
- Create a new pipeline job
- Pipeline Jenkinsfile as an example on the next slide
- Go to Jenkins > Manage Jenkins > Manage Credentials.
 - Add a Username/Password credential with your Docker Hub credentials (your-dockerhub-username and your-dockerhub-password)
 - Use docker-hub-credentials as the ID of the credentials
- Trigger the pipeline
- Browser: localhost:80

Jenkinsfile

```
• pipeline {
•   agent any

•   environment {
•     DOCKER_IMAGE = 'mynginx/webapp'
•     DOCKER_HUB_USERNAME = 'your-dockerhub-username'
•     DOCKER_HUB_PASSWORD = credentials('docker-hub-password-id') // Create Jenkins credentials for Docker Hub
•     GITHUB_REPO = 'https://github.com/your-username/your-repository.git'
•   }

•   stages {
•     stage('Clone Repository') {
•       steps {
•         git url: "${GITHUB_REPO}", branch: 'main'
•       }
•     }

•     stage('Build Docker Image') {
•       steps {
•         script {
•           // Build the NGINX Docker image
•           docker.build("${DOCKER_IMAGE}", "-f Dockerfile .")
•         }
•       }
•     }

•     stage('Push Docker Image to Docker Hub') {
•       steps {
•         script {
•           // Log in to Docker Hub
•           docker.withRegistry('https://index.docker.io/v1/', 'docker-hub-credentials') {
•             docker.image("${DOCKER_IMAGE}").push()
•           }
•         }
•       }
•     }

•     stage('Deploy to Docker') {
•       steps {
•         script {
•           // Stop and remove the existing container (if any)
•           sh 'docker rm -f webapp-container || true'

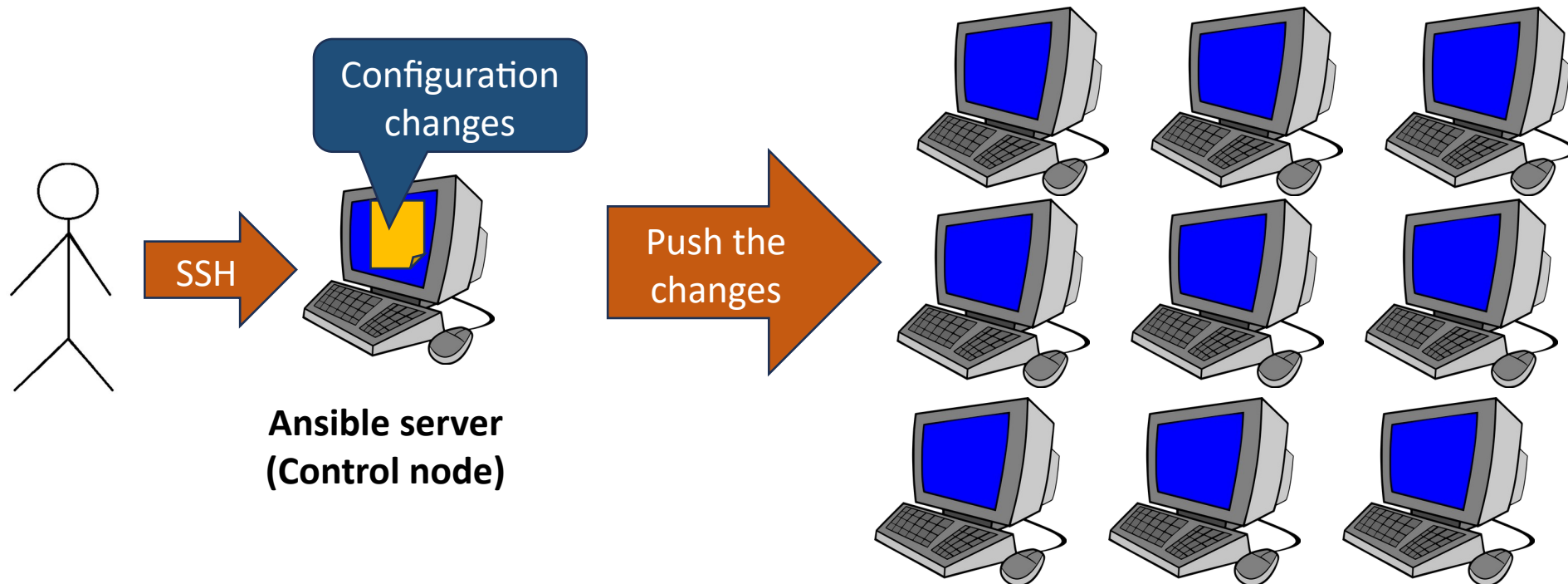
•           // Run the new Docker container
•           sh 'docker run -d -p 80:80 --name webapp-container ${DOCKER_IMAGE}'
•         }
•       }
•     }

•     post {
•       success {
•         echo 'Pipeline successfully executed.'
•       }
•       failure {
•         echo 'Pipeline failed.'
•       }
•     }
•   }
}
```

Ansible

Ansible

- **Ansible:** Free automation tool that can automate IT tasks on the local machine where it is running and on remote machines
- Installs on Linux and can configure Linux/Windows systems



Ansible Terminology

- Control node/Ansible server: Main server on which Ansible server runs
- Modules: Command to be executed on the client-side
- Task: Contains one or more modules
 - Example of a Task:
 - Install HTTP (Module)
 - Enable HTTP service (Module)
 - Start HTTP service (Module)
- Playbook: Automation file containing multiple tasks
- YAML: Format for creating playbooks
- Inventory: File containing information about remote clients where tasks are executed
- Tag: Alias/reference to a task
- Variable: Used for repetitive values
- Role: A small part of playbook

Ansible Usage

- Examples of ansible commands
 - Run modules from a YAML file: **ansible-playbook example.yml**
 - Run a module independently: **ansible myservers -m play**
- Ansible configuration files
 - /etc/ansible/ansible.cfg
 - /etc/ansible/hosts
 - /etc/ansible/roles

Ansible versus Chef/Puppet

Feature	Chef/Puppet	Ansible
Model	Pull model (Client->Server)	Push model (Server->Client)
We need to write code in	Ruby (Support is declining)	YAML
Agents	Need of agents	Agentless (Only SSH)
Installation	Relatively complex	Simple
Documentation	Average	Excellent

Ansible Installation

- Create two Ubuntu VMs (Server and Client)
- Both should have *Bridged adapter* as network interface
- On the server:
https://docs.ansible.com/ansible/latest/installation_guide/installation_distro_s.html
- Commands:
- **sudo apt update**
- **sudo apt install software-properties-common**
- **sudo add-apt-repository --yes --update ppa:ansible/ansible**
- **sudo apt install ansible**

Ansible Installation

- Check result: **ansible --version**
- If we see a *Ansible requires ... UTF-8* error:
 - `export LANG=en_US.UTF-8`
 - `export LC_ALL=en_US.UTF-8`
- To make these changes permanent:
 - `sudo nano ~/.bashrc`
 - `export LANG=en_US.UTF-8`
 - `export LC_ALL=en_US.UTF-8`
 - `source ~/.bashrc`
- Again try **ansible --version**
- Check connectivity between the local machine and ansible:

ansible localhost -m ping

(ansible CLI will check if it can connect to localhost using the *ping* module – It is not the ICMP ping, but it is Ansible's ping)

- Check whether configuration files are created: **ls /etc/ansible**

YAML File Syntax

- Do not use tabs, use spaces ... Indentation is very strict
- Start with **---** at the beginning of the file ... Then the file is considered as a YAML file ... Example

```
---
```

```
- name: sampleplaybook
```

```
  hosts: all or localhost
```

Where to run?

```
  become: yes
```

Run as a different user?

```
  become_user: root
```

```
  tasks:
```

```
    - name: Install Apache
```

```
      apt:
```

```
        name: httpd
```

Package name

```
        state: present
```

What to do? Install

Create a Playbook

- **mkdir /etc/ansible/playbooks** ... To keep our playbooks
- **cd /etc/ansible/playbooks** ... To keep our playbooks
- **sudo nano first.yml**
- **C:\lectures\CDAC\DITISS\ITIM\first.yml** ... Check if localhost can be reached
- Check syntax: **ansible-playbook --syntax-check first.yml**
- Dry run: **ansible-playbook --check first.yml**
- Run: **ansible-playbook first.yml**
- Deliberately make a syntax error and recheck

Running Ansible

- As a command: **ansible**
- With a playbook: **ansible-playbook**
- Example: Previous example using *ansible* command:
ansible localhost -m ping

More Examples

- **Output something:** helloworld.yml (The **debug** module is used to print something on the screen)
- **Run multiple tasks:** mtask.yml
- **Install Apache: packinstall.yml** (Platform-neutral version: packinstall-neutral.yml, Variable: variable.yml)
 - First check: **sudo systemctl status apache2**
 - Dry run: **ansible-playbook --check packinstall.yml**
 - Run: **ansible-playbook packinstall.yml**
 - Check status: **sudo systemctl status apache2**
 - Now manually remove apache: **sudo apt-get remove apache2**
 - Run: **ansible-playbook packinstall.yml**
 - Check status: **sudo systemctl status apache2**

Useful Ansible Modules

- **ansible.builtin.command**

- Directly executes a command/script on the target machine without invoking a shell
- The command is specified using **cmd** after this
- Example: Install Docker using *apt-get*

- **ansible.builtin.shell**

- Runs a shell command on the target machine
- As the command runs through a shell, feature such as piping, redirection, environment variables etc are allowed
- Example: *docker run* command needs shell access

Docker and Ansible

- `sudo mkdir /etc/ansible/playbooks/ansible_docker`
- `cd /etc/ansible/playbooks/ansible_docker`
- `sudo nano HelloWorld.java` ... Same file as before
- `sudo nano Dockerfile` ... Same file as before
- `sudo nano docker_java.yml`
- `ansible-playbook docker_java.yml --ask-become-pass`
- `docker ps -a`
- (Note: `--ask-become-pass` will ask for the user's password for sudo)

Running nginx using Docker With a Custom Page in Ansible

- `sudo mkdir /etc/ansible/playbooks/ansible_docker_nginx`
- `cd /etc/ansible/playbooks/ansible_docker_nginx`
- `sudo nano index.html` ... Add “Hello C-DAC message” here
- `sudo nano Dockerfile`
- `FROM nginx:latest`
- `COPY index.html /usr/share/nginx/html/index.html`
- `sudo nano nginx_docker_playbook.yml`
- `ansible-playbook nginx_docker_playbook.yml --ask-become-pass`

Remote Clients hosts File Syntax

- **Inventory:** Ansible calls all remote clients as inventory
- File */etc/ansible/hosts* contains inventory information – created during Ansible installation
- Example:
 - app1.example.com
 - app2.example.com
 - 206.10.23.89
- If we change the location of this file, use -i option
 - **ansible-playbook -i /home/cdac/ansible/hosts**
- To see the inventory: **ansible-inventory --list**

Changing Host Names

- It is better to assign clear host names to server and client machines
- **sudo nano /etc/hostname**
- Edit the existing host name to either **server** or **client**, depending on the machine, Ctrl+X, Y, Enter to save
- Refresh: **sudo systemctl restart systemd-hostnamed**
- Close and restart terminal ... The host name should have changed

Connect to a Remote Host

- **Client machine: Install SSH**
- **sudo apt-get update**
- **sudo apt-get install openssh-server -y**
- **sudo systemctl status ssh**
- **ip addr ...** Note the IP address (e.g. 192.168.0.187)
- **Server machine (Control node of Ansible)**
- **sudo nano /etc/ansible/hosts**
[labclients]
192.168.0.187
- Verify: **ansible-inventory --list**
- **ssh-keygen** and ENTER three times to generate key pair
- **ssh-copy-id 192.168.0.187 ...** Copy public key on the client machine

Connect to a Remote Host

- **Client machine**
- `cat ~/.ssh/authorized_keys` ... Verify that SSH key got copied
- `chmod 700 ~/.ssh` ... This and the next command to set file permissions
- `chmod 600 ~/.ssh/authorized_keys`
- `sudo nano /etc/ssh/sshd_config` ... Edit SSH configuration file
- `PubkeyAuthentication yes` ... Set correct permissions
- `PasswordAuthentication yes`
- `Ctrl+X Y Enter` ... Save the file
- `sudo systemctl reload sshd` ... Reload SSH configuration file
- **Server machine (Control node of Ansible)**
- `ssh -i ~/.ssh/id_rsa atul@192.168.0.187` ... Connect explicitly using key based authentication
- `hostname` ... Ensure that we see the client host name
- `exit` ... Exit SSH
- `ssh 192.168.0.187` ... Verify whether simple SSH works

Running Ansible Commands on the Client Through the Server

If we get permissions error after this:

Run this cmd on client machine:

```
cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
```

- On the Control node (Server)
 - **exit** ... Disconnect from the SSH
 - **ansible all -m ping** ... Ping all hosts mentioned in our Ansible hosts file
- If we see ping-pong from the client, it means Ansible is able to connect to the client from the server
- On the Control node (Server)
 - **ansible -a "uptime" all** ... To run the *uptime* command on the client directly
 - **ansible -a "ip a" all** ... See all client machines' IP addresses etc
- On the client
 - **cd ~**
 - **mkdir test**
- On the Control node
 - **ansible -a "ls -l /home/atul" all** ... Replace *atul* with your client user name, check whether you see the *test* directory in the output
 - **ansible -a "dpkg -l" all** ... See all packages installed on the client machines

Test Connectivity using a Playbook

- Test connectivity between the server and the client: **clientstatus.yml**
- **cd /etc/ansible/playbooks**
- **sudo nano clientstatus.yml**
- **ansible-playbook clientstatus.yml**

Copy Module: Copy a File to All Clients

- **echo "I am testing ansible" > /home/atul/testfile.txt**
- **Create /etc/ansible/playbooks/copy.yml**
- **ansible-playbook copy.yml**
- We might get permission error for sudo (Password missing)
- **sudo nano /etc/ansible/hosts**
- Edit the hosts entry to add client machine user and password like this:
- **[labclients]**
- **#192.168.1.9**
- **192.168.1.9 ansible_user=atul ansible_become_pass=atul**
- Save, exit and now retry: **ansible-playbook copy.yml**
- Verify: Client machine: **ls /tmp/testfile.txt** and then **cat /tmp/testfile.txt**

Change File Permissions

- Client: **cd ~/test** and then **touch one two three** and **ls -l**
- Server: Create **/etc/ansible/playbooks/filepermission.yml**
- **ansible-playbook filepermission.yml**
- Client: **ls -l**
- Permissions for file one should have changed
- How to change for all? We would need to use some advanced features

Setup Apache and Open Firewall Port

- Client: Ensure that Apache is not already installed: **systemctl status apache2**
- Server: Create **/etc/ansible/playbooks/httpsetup.yml**
- **ansible-playbook httpsetup.yml**
- Client: Ensure that Apache is installed and running: **systemctl status apache2**
- Open a browser on the Windows host machine/server: 192.168.0.164 (Or whatever is the client IP address) ... We should see Apache home page
- Note: If we face errors, restart both client and server machines, but that may result into IP address changes ... Update /etc/ansible/hosts file in that case and retry

Run Shell Scripts on a Remote Client

- Client: Create a shell script (**/home/atul/cfile.sh**)

#!/bin/bash

touch newfile

- Client: **chmod +x cfile.sh**
- Client: **ls -l** to verify that newfile does not exist
- Server: Create **/etc/ansible/playbooks/shellscript.yml**
- **ansible-playbook shellscript.yml**
- Client: **ls -l** to verify that newfile has been created now

Schedule a Job (Crontab)

- Server: Create **/etc/ansible/playbooks/cronjob.yml**
- **ansible-playbook cronjob.yml**
- Client: Verify if the job is scheduled: **crontab -u atul -l**
- Client: Verify newfile creation: **ls**
- Client: Terminate Crontab job: **sudo crontab -u atul e**
- Select nano ... Remove the crontab entry ... Save ... Exit
- **sudo rm newfile**
- Wait for a minute and again do **ls**, now newfile should not be created

User Account Management

- Server: Create **`/etc/ansible/playbooks/adduser.yml`**
- **`ansible-playbook adduser.yml`**
- Client: **`cat /etc/passwd`** to see if the new user has got created

Change Password

- Server: **sudo apt install python3-pip**
- Server: **pip install passlib**
- Server: Create **/etc/ansible/playbooks/changepass.yml**
- **ansible-playbook changepass.yml --extra-vars newpassword=abc123**
- Client: **sudo chage -l cdac** to verify last password change date and time

Roles

- **Role:** Simplifies a long playbook by grouping tasks into smaller playbooks
- Makes managing and sharing of playbooks easier
- We need to create a directory structure for this
- Example: On the right

- my_role/
 - └─ defaults/
 - └─ main.yml # Default variables
 - └─ files/
 - └─ file1.txt # Files to be copied to hosts
 - └─ handlers/
 - └─ main.yml # Handlers (e.g., restart a service)
 - └─ meta/
 - └─ main.yml # Metadata about the role (dependencies, etc.)
 - └─ tasks/
 - └─ main.yml # Main tasks for the role
 - └─ templates/
 - └─ config.j2 # Templates to be used in playbooks
 - └─ vars/
 - └─ main.yml # Variables specific to the role

Role Example

- Create a static inventory file listing multiple target nodes
- Develop a role to install and configure a database server (e.g., MySQL or PostgreSQL)
- Apply the role using an Ansible playbook to deploy the database server on one of the target nodes listed in the inventory

Step 1: Create Static Inventory File On the Server

- Create a new directory: `sudo mkdir ~/ansible_role_demo`
- Go to that directory: `cd ~/ansible_role_demo`
- `sudo nano inventory.ini`
- `# inventory.ini`
- `[db_servers]`
- `192.168.0.164 ansible_user=atul ansible_ssh_private_key_file=~/.ssh/id_rsa`
- `[db_servers:vars]`
- `ansible_python_interpreter=/usr/bin/python3`

Step 2: Develop a Role to Install and Configure a Database Server

- Step 2.1: Create the role directory structure (Single line command)
- `sudo mkdir -p ~/ansible_role_demo/roles/db_server/{tasks,handlers,files,templates,vars,defaults,meta}`
- Note: -p means create parent directories if they do not exist
- Step 2.2: Tasks for installing and configuring MySQL
- `sudo nano ~/ansible_role_demo/roles/db_server/tasks/main.yml`
- Step 2.3: Define default variables for the role
- `sudo nano ~/ansible_role_demo/roles/db_server/defaults/main.yml`
- Step 2.4: Define handlers to restart MySQL after configuration changes
- `sudo nano ~/ansible_role_demo/roles/db_server/handlers/main.yml`

Step 3: Create and Run the Playbook

- Server: **sudo nano ~/ansible_role_demo/site.yml**
- On the client machine: **sudo apt-get install python3-pip** and **sudo pip3 install PyMySQL**
- Back on the server: **ansible-playbook -i inventory.ini site.yml**
- On the client: **sudo systemctl status mysql**

In the Case of Errors ...

- Client machine:
- `sudo nano /root/.my.cnf`
- `[client]`
- `user=root`
- `password=root_password`
- `sudo chmod 600 /root/.my.cnf`
- `sudo chown root:root /root/.my.cnf`

In the Case of Errors ...

- Client machine:
- `mysql -u root`
- If it allows you to connect to MySQL prompt:
- `SELECT user, host, plugin FROM mysql.user WHERE user = 'root' AND host = 'localhost';`
- If this shows `auth_socket`, change it to password-based authentication:
- `ALTER USER 'root'@'localhost' IDENTIFIED WITH mysql_native_password BY 'root_password';`
- `FLUSH PRIVILEGES;`
- `exit`
- Now try going back: `mysql -u root`
- It should stop us, and we should have to use: `mysql -u root -p`

Terraform

- **Terraform**: Free and open source infrastructure management tool developed by Hashicorp
- Can deploy applications on any platform, single machine or Cloud
- Uses a declarative syntax language called **HCL** (Hashicorp Configuration Language)
- Files have a .tf extension
- Four-step process
 - Create .tf file
 - Init
 - Plan
 - Apply
- Works with resources (e.g. a machine, file, database, user)
- To install, visit <https://developer.hashicorp.com/terraform/install>
- Run the steps shown
- Check version: **terraform version**

HCL Basics

- Contains blocks and arguments in key-value formats

- Typical structure

```
<block> <parameters> {  
    key1 = value 1  
    key2 = value2  
}
```

- Block = Information about a platform and the resources we want to create on that platform

Terraform Example: Create a Local File

- **mkdir** ~/terraform-local-file
- **cd** ~/terraform-local-file
- **sudo nano local.tf**

```
<block> <parameters> {  
  key1 = value 1  
  key2 = value2  
}
```

```
resource "local_file" "languages" {  
  filename = "languages.txt"  
  content = "We love programming!"  
}
```

Block name

Resource type

local: Provider
file: Resource

Resource name

Arguments

- Note: Each resource type expects specific types of arguments
- Example: *local_file* will expect *filename*

Running Terraform: 4 Steps

- Create **.tf** file (Already done)
- Run **init** command: **terraform init**
 - Terraform will download and install a plug-in called *local*, since it knows that we want to create a local file, after reading *local.tf* file
- Run **plan** command: **terraform plan**
 - It will show us the steps for executing our terraform code
 - We should review this to ensure that it is doing what we want
- Run the **apply** command: **terraform apply**
 - It will wait for a response ... Type *yes*
- **terraform show**: View the details of the resource just created

Syntax Example: Create an AWS Instance

```
<block> <parameters> {  
  key1 = value 1  
  key2 = value2  
}
```

The diagram illustrates the syntax of a resource block with the following annotations:

- Block name:** Points to the `resource` keyword.
- Resource type:** Points to the string `"aws_instance"`.
- Resource name:** Points to the string `"webserver"`. Above this label, it specifies `local: Provider` and `file: Resource`.
- Arguments:** Points to the curly braces `{ }` enclosing the configuration lines.

```
resource "aws_instance" "webserver" {  
  ami = "ami-0c2f25c1f66a1ff4d"  
  instance_type = "t2.micro"  
}
```

Update and Destroy Infrastructure

- Suppose we want to remove all permissions for groups and others (i.e. make the file permission 700)
- Add the *file_permission* argument to *local.tf* file
- resource "local_file" "languages" {
 - filename = "languages.txt"
 - content = "We love programming!"
 - file_permission = "0700"
- }
- **terraform plan**
- **terraform apply**
- To delete the file: **terraform destroy**

Multiple Configuration Files in the Same Directory

- `sudo nano local.tf`
- `sudo nano networks.tf`
- `terraform init`
- `terraform plan`
- `terraform apply`

```
resource "local_file" "languages" {  
  filename = "languages.txt"  
  content = "We love programming!"  
}
```

```
resource "local_file" "networks" {  
  filename = "networks.txt"  
  content = "We love networking!"  
}
```

Multiple Configurations in One File

- `rm network.tf`
- `sudo nano local.tf`
- **`terraform init`**
- **`terraform plan`**
- **`terraform apply`**

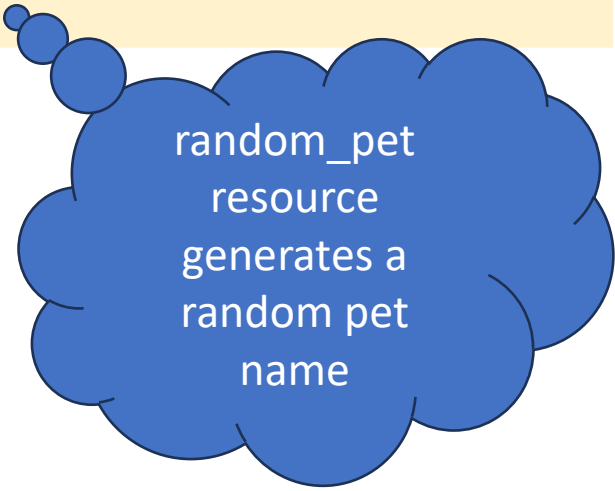
```
resource "local_file" "databases" {  
  filename = "databases.txt"  
  content = "We love databases!"  
}  
  
resource "local_file" "devops" {  
  filename = "devops.txt"  
  content = "We love devops!"  
}
```

- Note: The previously created text files (languages.txt and networks.txt) will be destroyed!

Multiple Providers in a Single Configuration File

- `cd ~`
- `mkdir terraform-multiple-providers`
- `cd terraform-multiple-providers`
- `sudo nano main.tf`
- `terraform init`
- `terraform plan`
- `terraform apply`
- Result: Pet name in the output, not in pets.txt file

```
resource "local_file" "pet" {  
  filename = "pets.txt"  
  content = "We love pets!"  
}  
  
resource "random_pet" "my-pet" {  
  prefix = "Mrs"  
  separator = "."  
  length = "1"  
}
```



random_pet
resource
generates a
random pet
name

Using Input Variables

- **sudo nano main.tf**
- **sudo nano variables.tf**
- **terraform init**
- **terraform plan**
- **terraform apply**
- Then change one variable value
- Rerun

```
resource "local_file" "pet" {  
    filename = var.file_name  
    content = var.content  
}  
  
resource "random_pet" "my-pet" {  
    prefix = var.prefix  
    separator = var.separator  
    length = var.length  
}
```

main.tf

```
variable "file_name" {  
    default = "pets.txt"  
}  
  
variable "content" {  
    default = "We love pets!"  
}  
  
variable "prefix" {  
    default = "Mrs!"  
}  
  
variable "separator" {  
    default = "."  
}  
  
variable "length" {  
    default = "1"  
}
```

variables.tf

Terraform and AWS: AWS IAM

- AWS account: Our main AWS account is **root account** (Has all the permissions)
- Ideal: We should not use it for regular work, but create other users using root
- Other users can either have
 - Management console access (Needs user id and password) OR
 - Command-line access (Needs access key id and secret access key)
- Use the **IAM (Identity and Access Management)** module to set permissions
 - Example: *AdministratorAccess* policy will allow all operations
- What about services within AWS (e.g. an EC2 instance needs access to an S3 bucket)
 - Not possible directly - We need to create a role, attach policies and assign the EC2 instance to that role
 - Example: Create a role *my-s3-access*, attach policy *AmazonS3FullAccess* to it and assign our EC2 instance that role

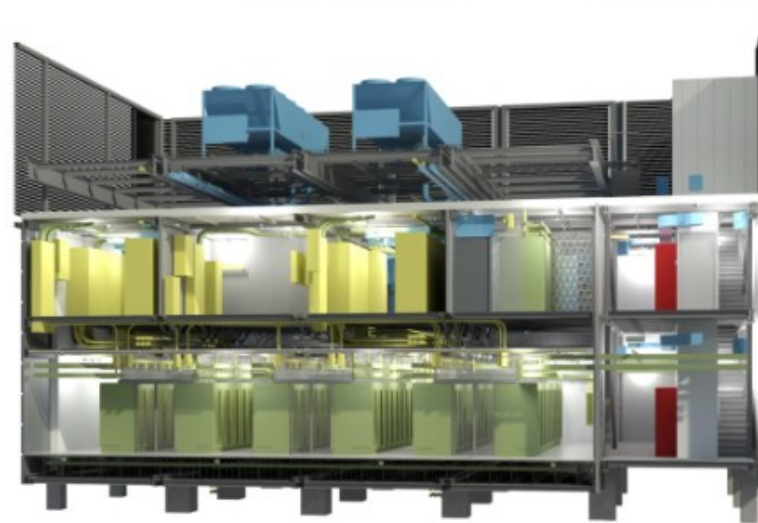
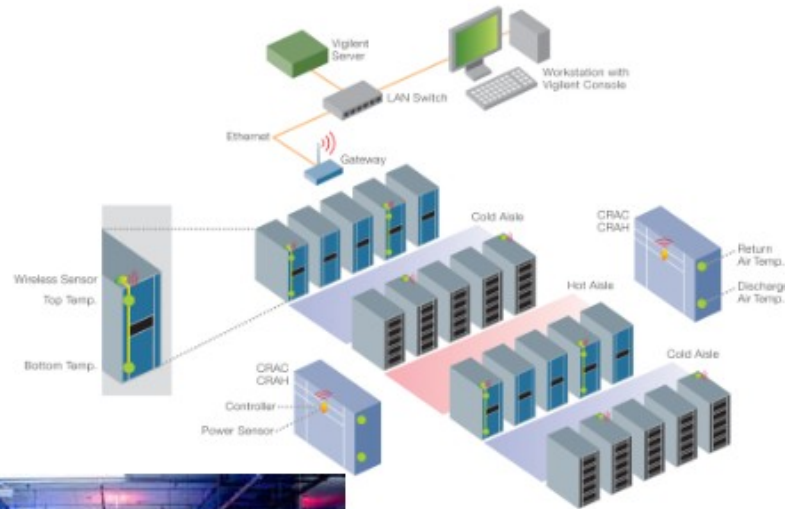
Data Center Management

Ref: Administering Data Centers book by Kailash Jaiaswal

What is a Data Center?

What is a Data Center?

- A place that houses computer systems
- Networks, storage, processing, etc.



Data Center Pre-requisites – Physical Area

- Physical capacity is defined as available space for servers, storage, network devices, HVAC (heating, ventilation, and air conditioning), power panels, breakers, and floor to support the weight of the equipment

Data Center Pre-requisites – Power

- To protect against power failures, uninterruptible power supplies (UPSs) must be present in every data center
- If the utility company has frequent long-duration outages, generators are required

Data Center Pre-requisites – Cooling and HVAC

- HVAC is required to keep the devices cool and maintain low humidity within the data center
- Like power, the HVAC system is very difficult to retrofit
- Therefore, the HVAC system must have enough cooling capacity to meet present and future needs
- Cooling requirements are measured in British thermal units (BTUs) per hour
- The HVAC manufacturer provides this
- What you must know is the combined BTU-per-hour requirement for all of the equipment in the data center

Data Center Pre-requisites – Required Weight

- It is critical to know the weight of each rack without any content (server, storage device) inside it and the combined weight of all the racks
- Add the weight of all devices (approximate estimate) that will be placed in the racks
- That will give you an idea of the weight that the data-center floor will be subjected to
- You can estimate if the existing floor is strong enough for the weight
- If it is not, you must make suitable changes

Data Center Pre-requisites – Required Weight

- It is critical to know the weight of each rack without any content (server, storage device) inside it and the combined weight of all the racks
- Add the weight of all devices (approximate estimate) that will be placed in the racks
- That will give you an idea of the weight that the data-center floor will be subjected to
- You can estimate if the existing floor is strong enough for the weight
- If it is not, you must make suitable changes
- Types of load to be considered:
 - Maximum weight the entire data center floor can support
 - Maximum weight a single tile can support
 - Maximum point load a tile can support

Data Center Pre-requisites – Network Bandwidth

- The bandwidth offered by the Internet service provider (ISP) should at least be equal to the data center's inbound and outbound bandwidth specifications
- Since the cost of bandwidth, network switches, and ports goes down over time, it is worthwhile to provision only the amount required and add more if and when needed
- If business-critical servers must be connected to the Internet, reliable and redundant ISP links to the Internet are mandatory

Data Center Pre-requisites – Budget Constraints

- In theory, we can get more than enough money for constructing a data center, but in practice, this rarely or never happens
- The toughest thing about designing and constructing a data center is staying within the allocated budget
- We will have to make certain compromises, such as not having generators or redundant HVAC units
- Each compromise reduces the cost but adds some risk, and we must weigh the risk of each decision
- We can also look for features or areas that can be added or upgraded later

Selecting a Geographic Location

- Should be safe from natural hazards such as floods, fires, tsunamis, earthquakes, etc
- Should be safe from human-made disasters such as avoiding places near airports, telecommunication centres to avoid RF interference and also near highways, mines, quarries to avoid vibrations
- Should be close to locally available talent pool
- Should have abundant and inexpensive utilities such as power and water

Selecting an Existing Building (Retrofitting)

- The data center should be located on the ground or first floor of the building
- There should be enough parking and entrance/exit space for ambulance, fire department vehicles, large trucks
- It should have a large docking/unloading area
- Setting up network cables should be easy
- Enough power and backup power should be available for 24x7 operation
- Gas and water pipes must be leak free

Key Design Characteristics of a Data Center

- Simple
- Scalable
- Modular
- Flexible

Guidelines for Planning a Data Center

- Plan in advance
- Plan for the worst
- Plan for growth
- Simplify the design
- Plan for changes
- Label all the equipment , especially cables and ports

Data Center Structure

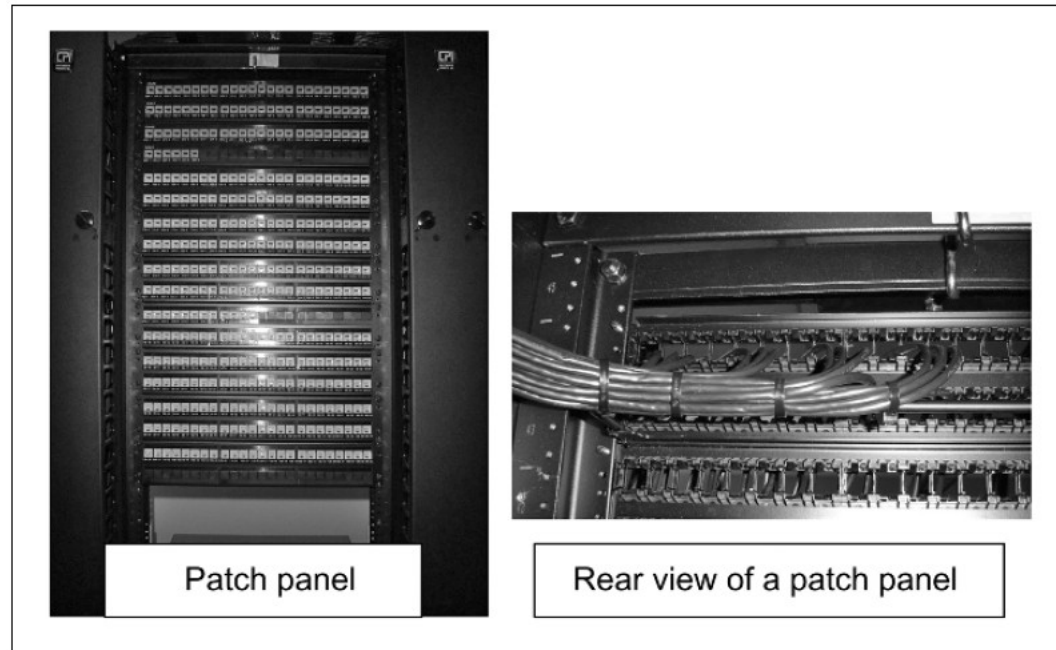
- No-raised floor
 - To prevent someone from removing a tile and accessing data center resources illegally
 - Allows separate compartments for different customers for better security
- Aisles
 - The space between two rows of racks and also at the corners should be enough for removing old racks and moving in new racks
- Ramp
 - Makes it easy to move equipment in or to take it out

Design and Plan Against Vandalism

- The building should make it easy to control access
- Check for existing doors, windows, ventilators that open outside and to uncontrolled areas
- Place CCTV
- Install alarm systems and motion detectors
- The data center should be in the interior part of the building to make it difficult to access from the outside
- Data center should have one/two points of access
- If the data center holds data and applications of different companies, their access should be physically separated and controlled
- Make provision for emergencies, have fire extinguishers etc
- Strictly need-based access should be provided and recorded

Network Infrastructure – Data Cabling Design

- Modular cabling design using *patch panels*
 - All hosts and other devices that need network connections are cabled to ports in a patch panel located close by in the data center



Network Infrastructure – Points of Distribution (POD)

- A point of distribution (POD) is a rack with devices, network switches, terminal servers, and patch panels that together provide the infrastructure required for a number of racks filled with servers and storage
- PODs allow you to group cables and networking equipment required for a set of racks into a single manageable location

Network Infrastructure – ISP Network Infrastructure

- Work with a reliable vendor for routers, switches, etc
- If there are multiple vendors, their equipment should be cross-compatible

Network Infrastructure – ISP WAN Links

- A WAN link has two parts: Transport and transit
- Transport is the path which has a maximum bandwidth (*water pipe*), transit is the actual data (*water*)
- The WAN link should have sufficient bandwidth
- A backup line should be available for dealing with main link problems
- Sufficient redundancy should be implemented

Network Operations Center (NOC)

- The network operations center (NOC) is a dedicated facility staffed with people (usually at all hours) who monitor the availability of all devices and services within the data center and respond to any data-center problems
- The NOC has servers, consoles, and network monitoring software

Network Monitoring

- Network monitoring is knowing what is happening within your environment
- Network monitoring gathers real-time data and classifies it into performance issues and outages

Data Center Physical Security

- All entry points and doors to the data center must be controlled by card readers or persons who are physically present to monitor and restrict access of those entering the data center
- Biometric devices can be used
- Cameras must be installed not only at the data-center doors but also at the corporate entrances
- Badge access to the data center must not be given to all employees
- Only those who must access the data center must be allowed to enter

Data Center Logical Security

- Logical security is making it more difficult for intruders to reach a login prompt on the hosts or other devices
- The telnet port should be closed
- Use ssh instead to log in to UNIX servers
- Protocols using low-number ports (less than 1,024) should be allowed only if necessary
- You must construct more than one layer of authentication before presenting the login prompt and must also allow only a few and necessary users the capability to go beyond these layers
- These users must be forced to authenticate themselves to a central login server, which should be the only machine having direct access to the consoles

Data Center Cleaning

- A data-center setting is different from an office area and, therefore, does not require regular cleaning or vacuuming
- You must get a cleaning crew in the data center once every 3 or 6 months or as needed, such as after structural (floor, tile, wall, and so forth) changes
- The cleaning crew must be given a data-center map that designates electrical outlets they can use
- They must know and follow all rules: no food or drink in the data center, no interfering with ongoing operations, no leaving doors propped open, and no unbadged/unauthorized personnel

Reasons for Data Center Consolidation

- Reduce number of servers
- Increase usage of storage
- Reduce IT processes
- Reduce support staff
- Reduce IT expenses
- Increase service availability

Consolidation Opportunities

- Server consolidation
 - Vertical scaling specifies how you can reduce the server count by running multiple applications on fewer but more powerful servers
 - Horizontal scaling allows you to bear increased workload by distributing it across several replicated servers
- Storage consolidation
 - Consolidate servers
 - Link servers to a single storage server
 - Storage Area Network (SAN) – Single point of access for all storage
 - Network Attached Storage (NAS) – Example: Access using TCP/IP

Consolidation Opportunities

- Network consolidation
 - Maintain a diverse network and consolidate staff
 - Select existing good networks to keep them, link them to new, better networks
- Application consolidation
 - Database instances
 - Applications
- Service consolidation
 - SSO
 - LDAP
- Process consolidation
 - Document and define booting, backup, authentication etc

Data Center Consolidation Phases

- Phase 1: Study and document the current environment
 - Evaluate application requirements, hardware infrastructure, storage requirements, networking requirements, operations requirements, risk
- Phase 2: Architect the target consolidated environment
 - Analyze the collected data, List architectural requirements, Create an initial architecture, Size all equipment, Create a prototype, Test it, Revised the proposed architecture, Document it
- Phase 3: Implement the new architecture
 - Draft all low-level specifications, Construct the new environment, Create a data migration process, Back up data in the old environment, Migrate data and applications to the new environment, Train the operations staff
- Phase 4: Control and administer the consolidated environment
 - Increase accountability, availability, security

Server Capacity Planning

- Server sizing is an estimate of hardware requirements based on applications, anticipated activity, and satisfactory performance levels
- For example, to meet the needs of 3,000 end users, you require two load-balanced application servers with four 1.3 GHz CPUs and 8 GB of RAM, and one back-end database server with 8 CPUs and 10 GB of RAM
- Main points
 - Identify the slowest link, Define customer requirements, Estimate current resource utilization, Size the new servers

Disaster Recovery

- Five phases:
 - Phase 1: Find the Funds
 - Phase 2: Assess Existing Environment and Risks
 - Phase 3: Create DR Procedures
 - Phase 4: Test the Procedures
 - Phase 5: Adjust Your DR Plan to Site Changes and Technical Advancements

Data Center Security Guidelines

- Keep OS and applications updated
- Install and update antivirus software
- Don't open or use unexpected attachments
- Use secure programs
- Install firewalls and customize them—
- Use good passwords

Internet Security Guidelines – 1

- Minimize the operating system. Install necessary packages or software sets
- Place the server in a DMZ, and set the firewall to drop connections on all but the required ports
- Harden the OS
- Keep up-to-date on patches, especially those related to Internet security
- Install firewalls, IDS/IPS at all appropriate stations - These are devices that inspect incoming network traffic and decide whether they
- should go through or not, based on source, destination, data type, and other factors
- Deploy security applications that detect attacks and send immediate alerts if necessary
- Remove all services from the host except those that are required

Internet Security Guidelines – 2

- Review startup scripts and ensure that unnecessary daemons are not started at boot time
- Permit administrative logins from a few trusted internal systems only
- Limit the number of people who have accounts on the server - Only a few people need to know the root password
- Disable all remote administration unless it is done over an encrypted link or using a one-time password
- Log all user activity and keep an encrypted copy of the log files on the local system or another server on the network
- Monitor logs for suspicious activity. Configure automatic alerts if certain messages are logged or critical files (such as user account or application configuration files) are modified

Internet Security Guidelines – 3

- For Internet application software such as web servers, remove all “default” directories that are shipped with the Web server software
- Scan the external servers periodically using tools such as nmap and Internet security systems (ISS) to look for vulnerabilities
- Configure intrusion-detection software to monitor all client connections and send an alert when it sees any suspicious activity or log
- Stay up-to-date

Best System Administration Practices

- Pay attention to details
- Document
- Check simple and obvious things
- Keep the users informed
- Test
- Plan for delays
- Fix root cause of the problem
- Automate as much as possible
- Create backups
- Do not try new things in production environment

System Administration Automation

- Software configuration tracker
- Log file scanner
- Usage tracker
- User account management