NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

Cloud Migration and Microservices

Cloud Infrastructure Engineering

Nanyang Technological University
& Skills Union - 2022/2023

Recap

If class agree, we can spend the first 1 hour or more to recap and redo prev. lesson since today's lesson is not a technical lesson

Course Content

- Self Study Check In
- Introduction to Cloud Migration
- Key Benefits of Cloud Migration
- Pros & Cons of Serverless
- Activity

Q1: What is the primary benefit of cloud migration?
a) Lower security risks
b) Reduced operational costs
c) Limited scalability options
d) Increased physical hardware maintenance

Q2: Which of the following cloud migration strategies involves moving an application from an on-premises environment to the cloud with minimal changes to its architecture?

A) Lift-and-shift
B) Refactoring
C) Replatforming
D) Repurchasing

Q3: Which of the following is a potential risk in cloud migration?
a) Increased data accessibility and security
b) Reduced flexibility and scalability
c) Lower migration costs compared to on-premises solutions
d) Vendor lock-in to a cloud provider

Q4: Which cloud migration strategy involves replacing an existing application with a commercial off-the-shelf (COTS) software or a Software as a Service (SaaS) offering?

a) Rehosting
b) Refactoring
c) Repurchasing
d) Retiring

Activity

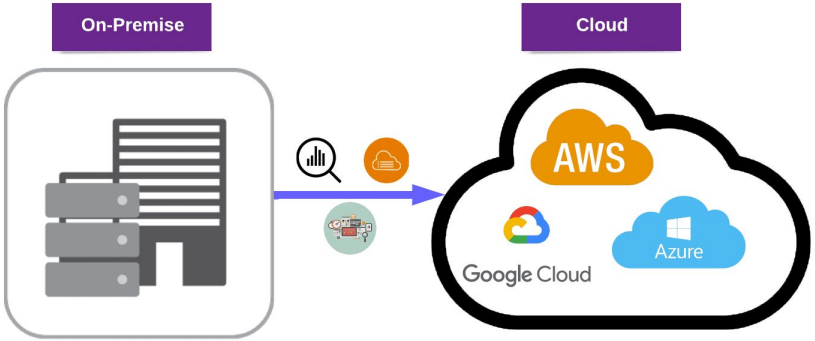
Instructor

- Ask to use AWS use single region for all learner for easier monitoring

Cloud Migration

What is Cloud Migration?

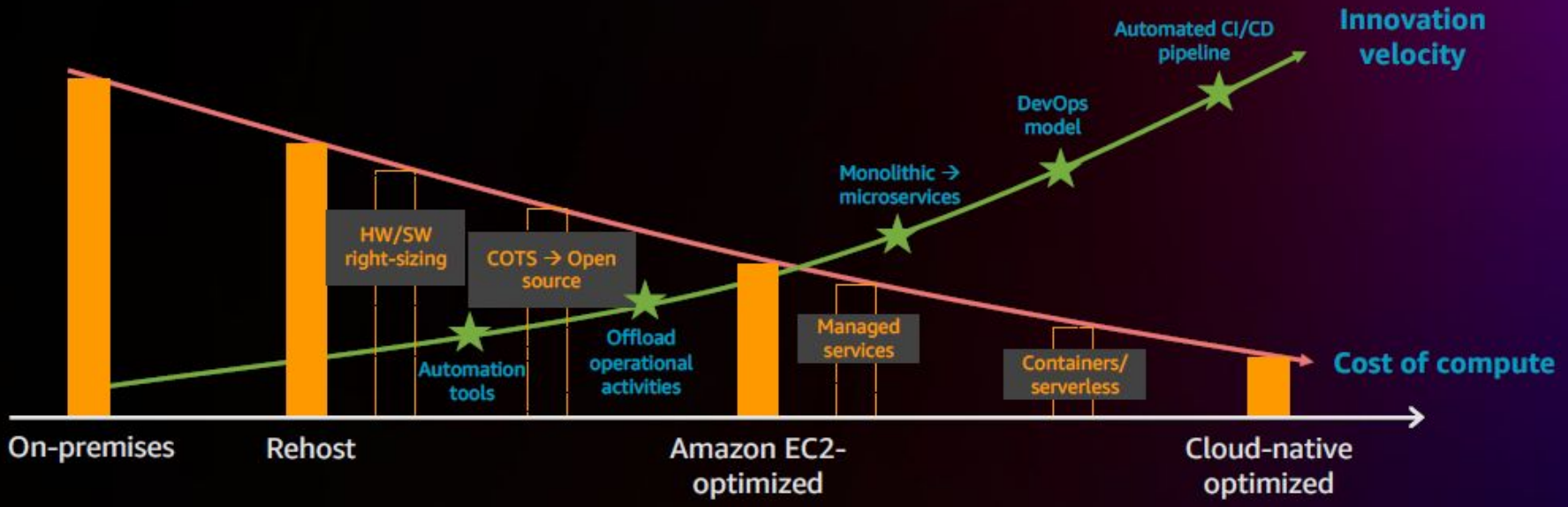
Cloud migration is **the process of moving data, applications, and other digital assets** from an organization's **on-premises infrastructure to cloud** computing environments or **from one cloud environment to another**



Why are we migrating to the cloud?

- To reduce your operating cost through pay-as-you-go pricing
- Improve your workload scalability without investing in new hardware.
- Establish Better Disaster Recovery Capabilities
- Make workloads highly available
  - using Multi-AZ
- Improve Agility
- Improve Scalability
- Data Center Decommissioning
- Lower TCO(Total Cost of Ownership)
- Faster Time to market

Migration and modernization journey



Cloud Migration Phases

1) Assess and Plan
2) POC/ Pilot Migration
3) Data Migration
4) Application Migration
5) Leverage the Cloud
6) Cloud Optimization

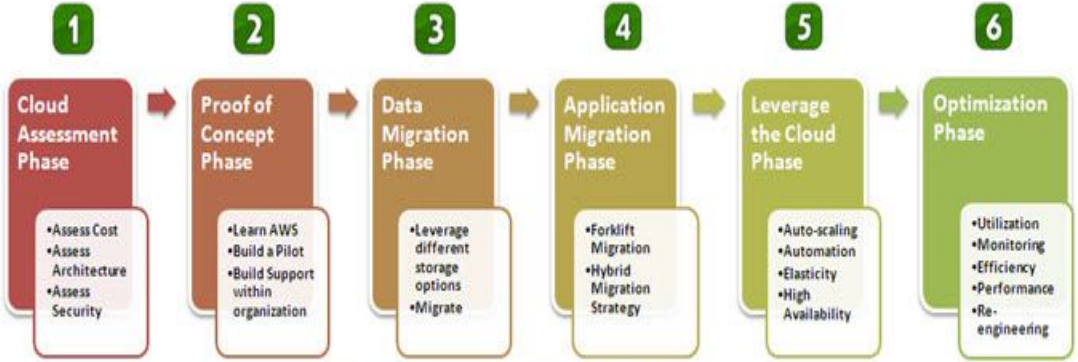


Figure 1: A Phase Driven Approach to Cloud Migration

http://media.amazonwebservices.com/CloudMigration-main.pdf

amazon web services

Understand key business drivers, scope, and timeline of Cloud Migration

- Why you want to migrate?
- What are the key business drivers?
- What is in scope and out of scope?
- What are the compelling events/milestones?

What are we migrating to Cloud?

1st Example:- Migrating a single application:

- A small business moves its customer relationship management (CRM) system from an on-premises server to a cloud-based SaaS solution like Salesforce or Microsoft Dynamics 365.

What are we migrating to Cloud?

2nd Example:: Migrating an entire data center:

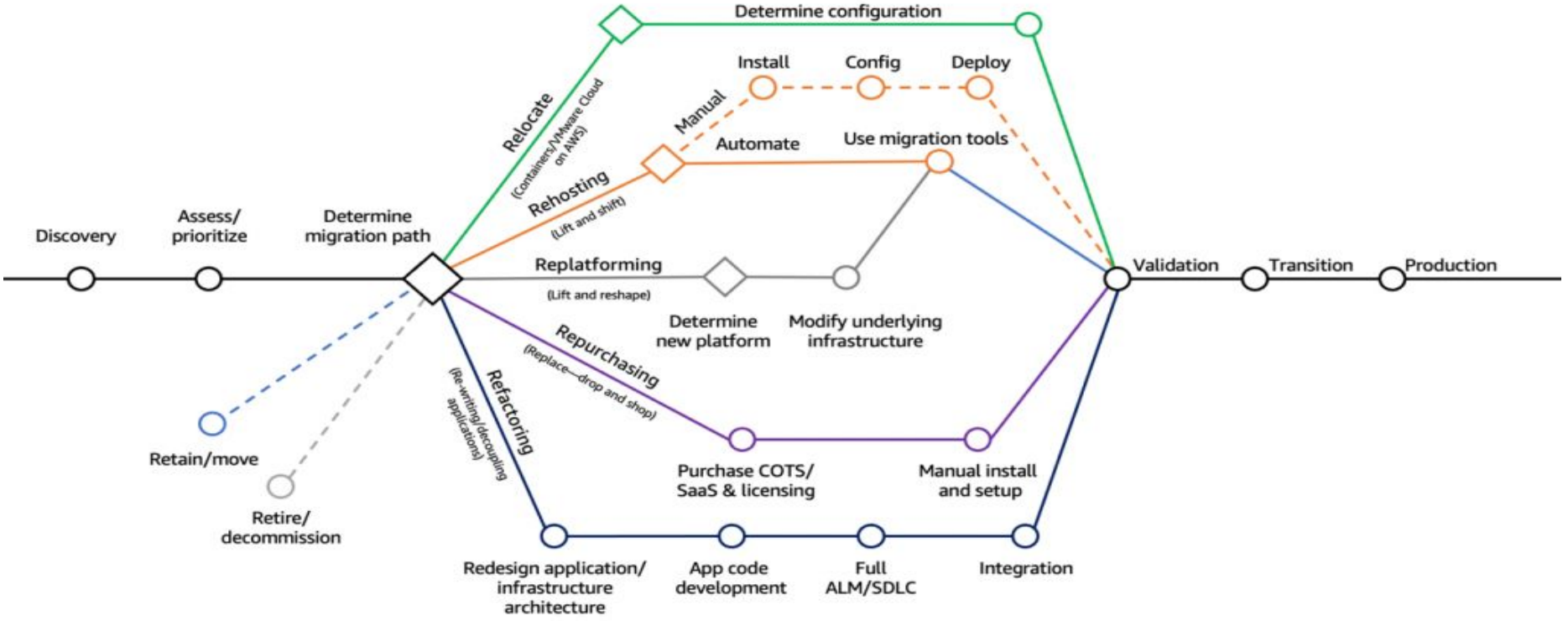
- A large organization moves its entire on-premises data center, including applications, databases, storage, networking, and security infrastructure, to a cloud provider like AWS. (This could include thousands of servers, Petabytes of data)

  Note:- Scope may change in Large migrations due to Unknowns

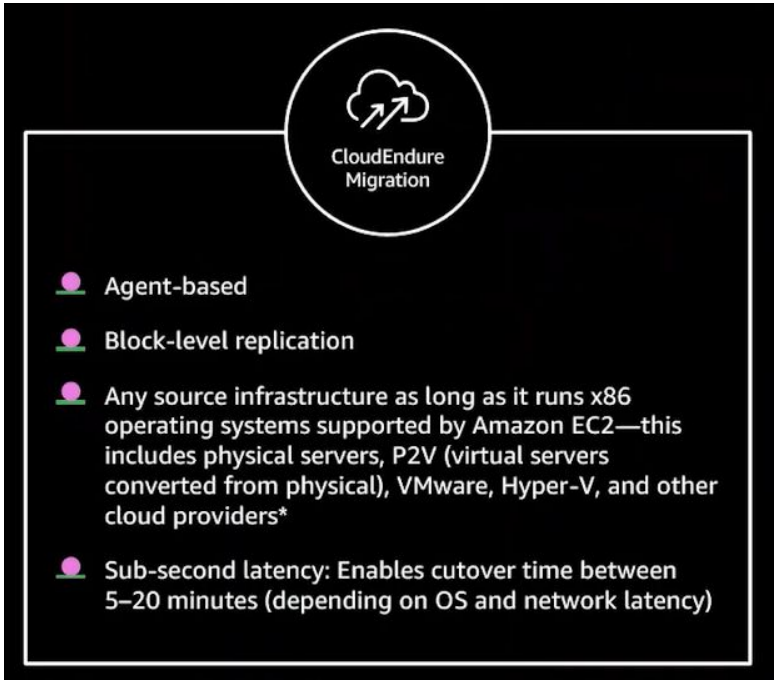
Build a migration strategy

- Build a 7 Rs strategy for your assets
  - Eg:- 70 % rehost, 20 % replatform, 10 % refactor

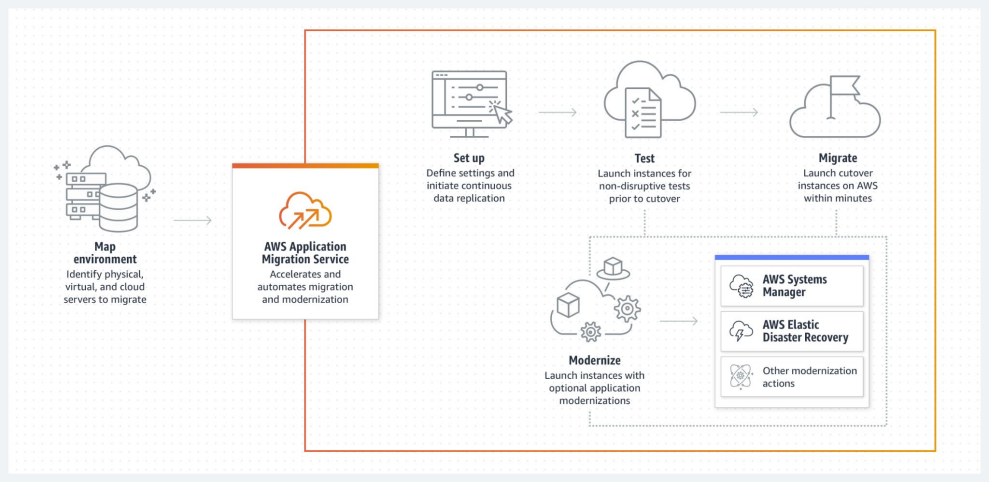
7R of Migration Strategies



Rehost(Lift and Shift)



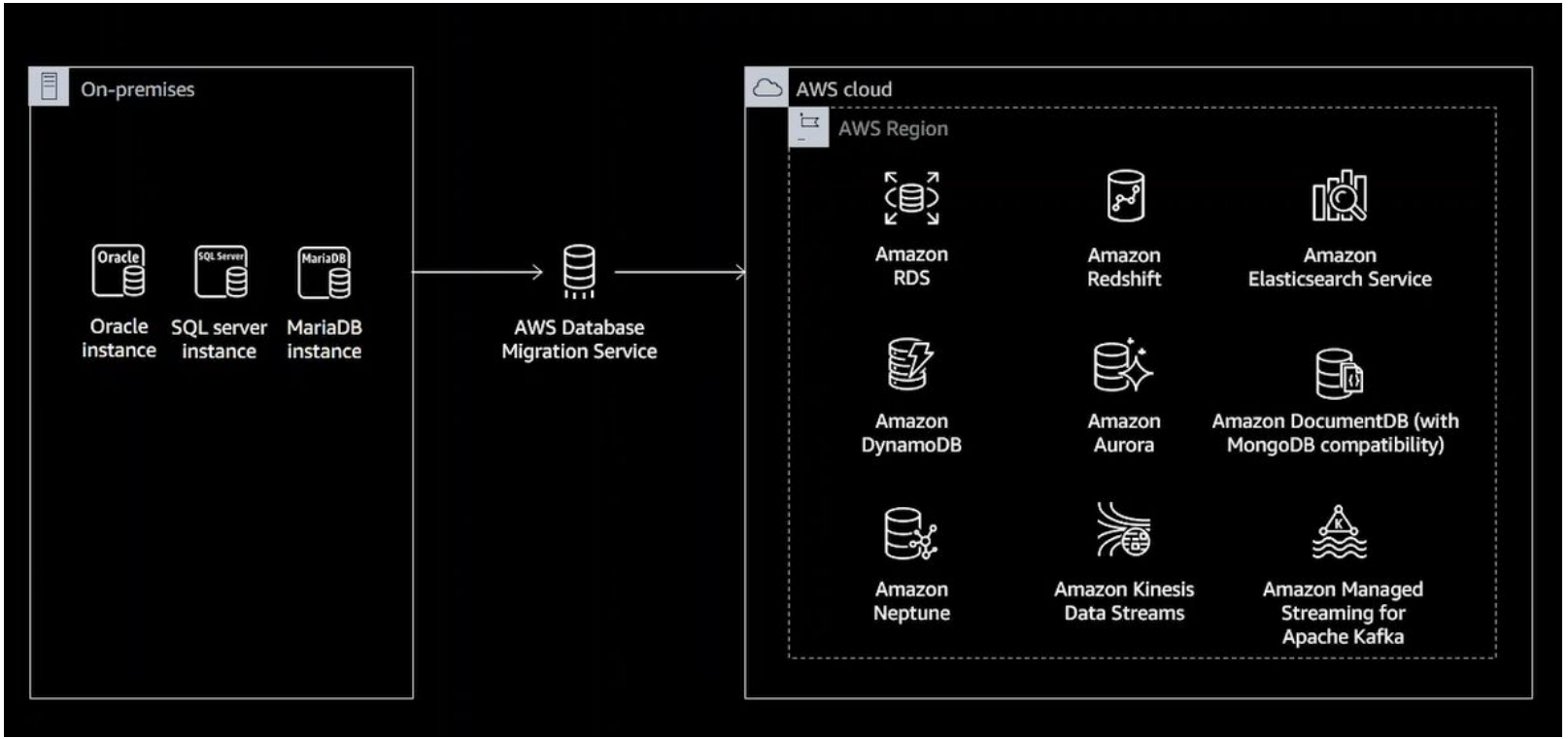
CloudEndure(Proprietary)



AWS Application Migration Service

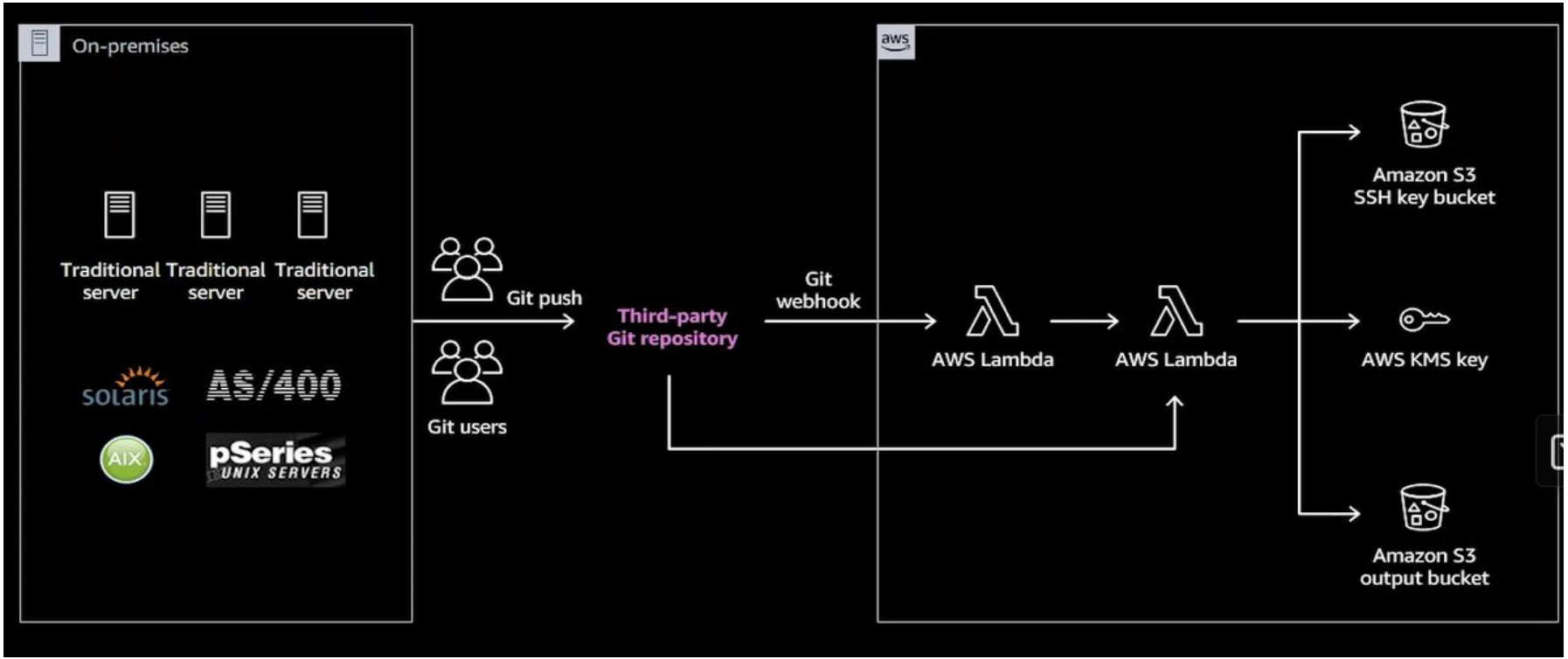
https://d1.awsstatic.com/cloudendure-migration-and-aws-application-migration-service-comparison-table.pdf

Replatform


On-premises
Oracle
SQL Server
MariaDB
Oracle instance
SQL server instance
MariaDB instance
AWS Database Migration Service
AWS cloud
AWS Region
Amazon RDS
Amazon Redshift
Amazon Elasticsearch Service
Amazon DynamoDB
Amazon Aurora
Amazon DocumentDB (with MongoDB compatibility)
Amazon Neptune
Amazon Kinesis Data Streams
Amazon Managed Streaming for Apache Kafka

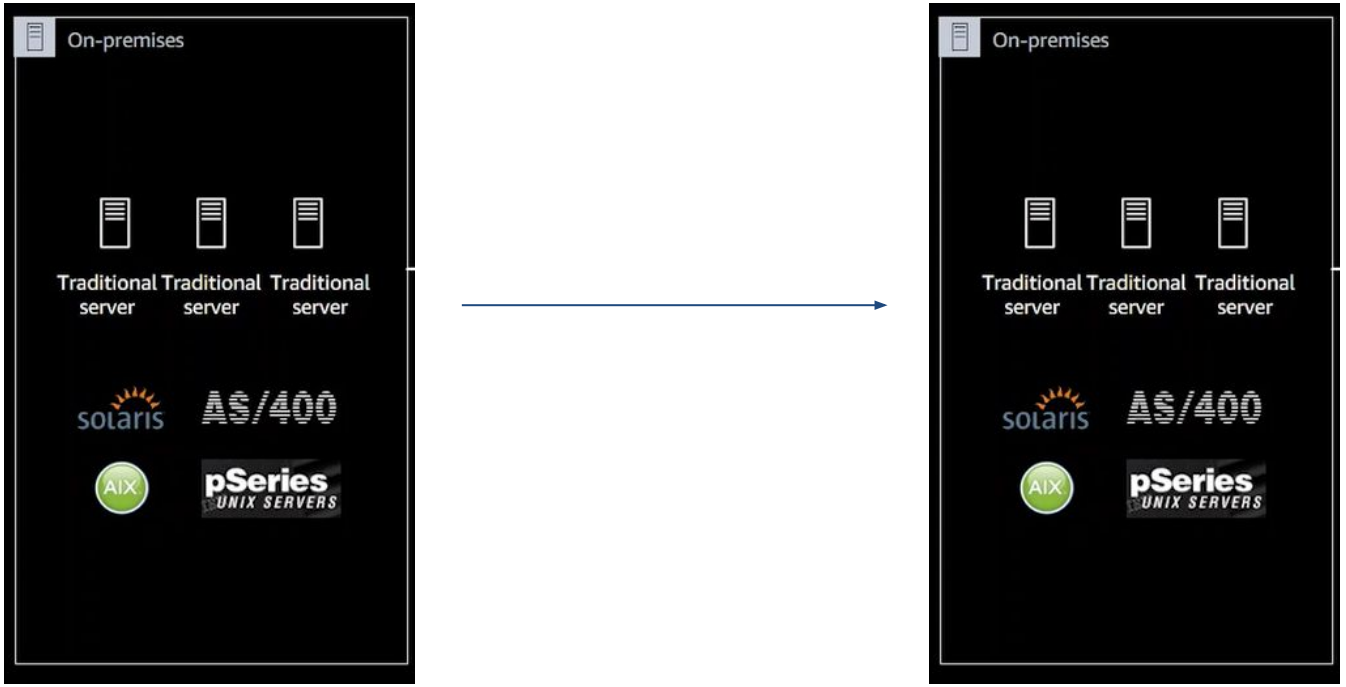
Refactoring

Re-imagining how the application is architected and developed, typically using cloud-native features.



Retain

Usually this means “revisit” or do nothing (for now)



Before Migration

After Migration

Repurchase

”Moving to a different product”

Eg:- Outdated on-premise CRM software → SalesForce Cloud

- Reduce effort/increase speed of migration
- Reduce in-house skills requirements

https://aws.amazon.com/marketplace

Relocate

[Hosted VMware – VMware Cloud on AWS – Amazon Web Services]

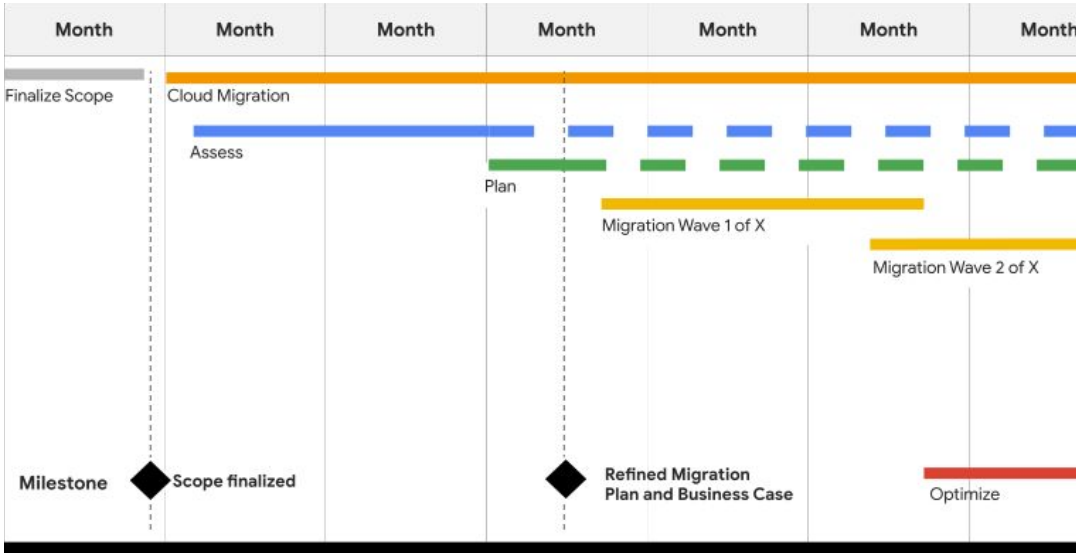
Retire

This is the migration strategy for the applications that you want to decommission or archive. Retiring the application means that you can shut down the servers within that application stack. The following are common use cases for the retire strategy:

- There is no business value in retaining the application or moving it to cloud.
- You want to eliminate the cost of maintaining and hosting the application.
- There has been no inbound connection to the application for the last 90 days.

Create a Migration Plan and Timeline

- When do you need to complete the Migration?
- What are the milestones?



Cloud Migration Best Practices

- Planning for gradual rollout and deployments
- Ensuring that you have a rollback strategy for each step of the migration plan
- Automate manual tasks to save time and costs
- Leverage managed services(e.g: use RDS)
- Establish accountable leaders & train your teams
- Monitor application performance
- Validate cloud security
- Identify key stakeholders for each activity and plan communications

Microservices

Microservices are an architectural and organizational approach to software development where software is composed of **small independent services that communicate over well-defined APIs**. These **services are owned by small, self-contained teams**.

Microservices architectures make applications easier to scale and faster to develop, enabling innovation and accelerating time-to-market for new features.

Monolithic vs Microservices

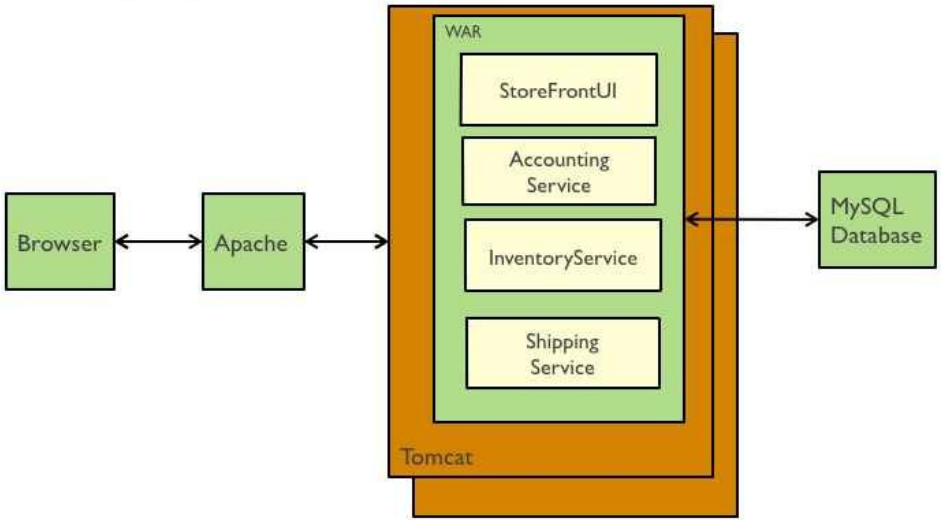
Monolithic architectures,

- Tightly coupled
- Entire application must be deployed and scaled.
- Adding or improving a monolithic application features becomes more complex as the code base grows. This complexity limits experimentation and makes it difficult to implement new ideas.
- Monolithic architectures add risk for application availability(single point of failure)
- Testing is easy

Microservices architecture,

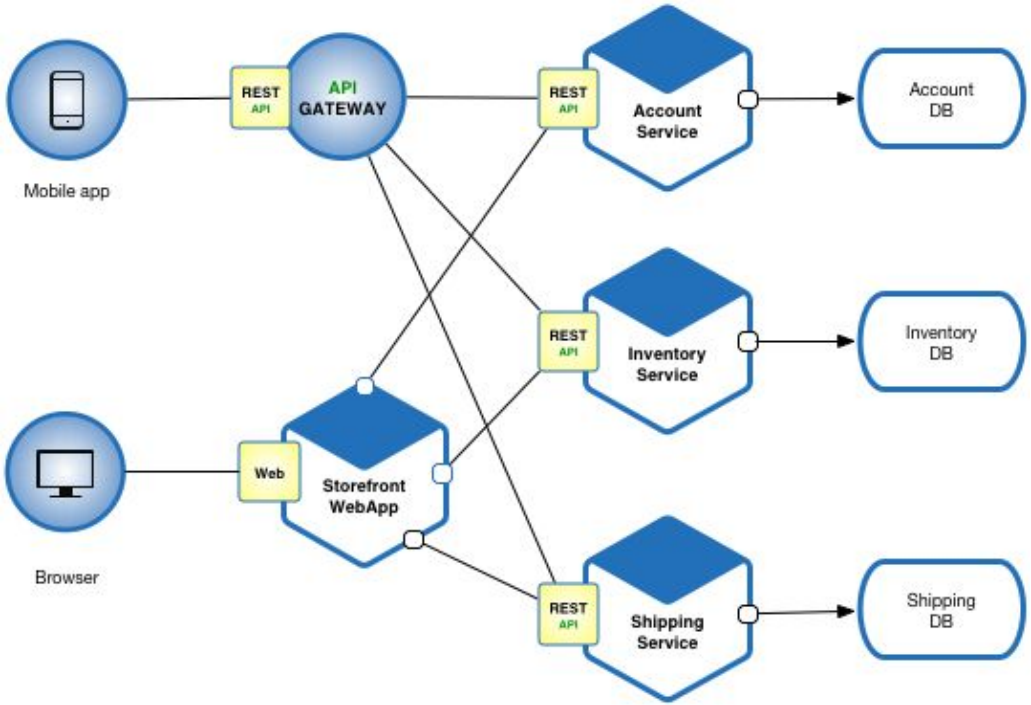
- An application is built as independent components that run each application process as a service.
- Flexible Scaling- Per service
- Easy deployment
- Code base is small per service. Therefore easier to understand and experiment.
- High levels of agility
- Testing is harder due to since there are so many independently deployable components.
- Technological Freedom
- Reusability of the service

Monolithic Architecture



Eg:- E Commerce Application

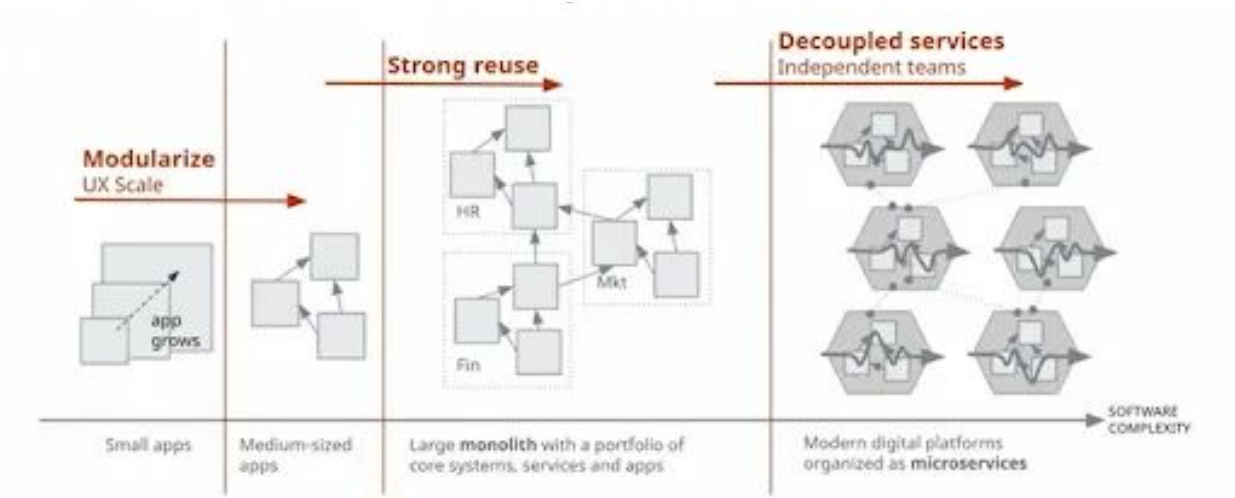
Microservices Architecture



Eg:- E Commerce Application

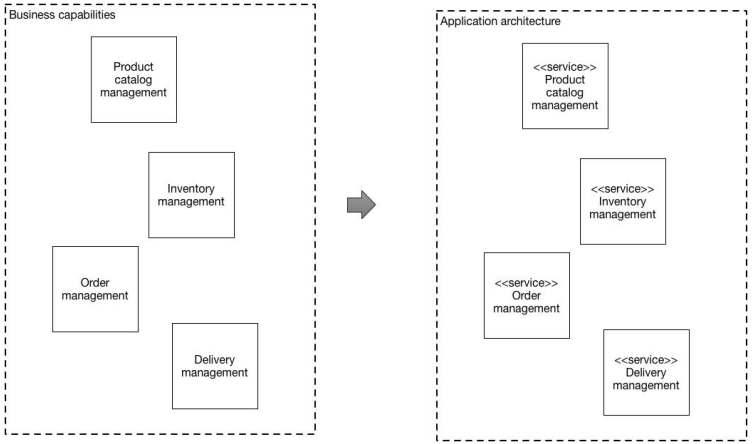
When to use Microservices?

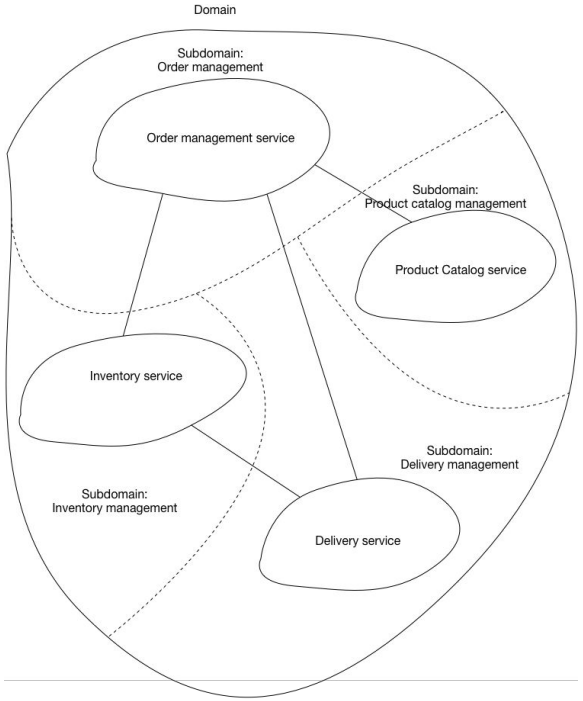
- When your current enterprise architecture requires modularity
- When your software application has to embrace container-based deployment
- When software complexity increases



Decompose

- Decompose by business capability
- Decompose by subdomain





Microservices - Deployment Patterns

- Service instance per Container - deploy each service instance in its container(eg:- Docker)
- Serverless deployment - deploy a service using serverless deployment platform(eg:- Lambda)
- Service deployment platform - deploy services using a highly automated deployment platform that provides a service abstraction (eg:- Kubernetes/ECS)
- Service instance per host - deploy each service instance in its own host(Eg:- Bare metal)
- Service instance per VM - deploy each service instance in its VM (Eg:- EC2)
- Multiple service instances per host - deploy multiple service instances on a single host(eg:- OSGI bundles)

Characteristics of Microservices

- Autonomous

Each component service in a microservices architecture can be developed, deployed, operated, and scaled without affecting the functioning of other services. Services do not need to share any of their code or implementation with other services. Any communication between individual components happens via well-defined APIs.

- Specialized

Each service is designed for a set of capabilities and focuses on solving a specific problem. If developers contribute more code to a service over time and the service becomes complex, it can be broken into smaller services.

Benefits of Microservices

- Agility

Microservices foster an organization of small, independent teams that take ownership of their services. Teams act within a small and well understood context, and are empowered to work more independently and more quickly. This shortens development cycle times. You benefit significantly from the aggregate throughput of the organization.

- Flexible Scaling

Microservices allow each service to be independently scaled to meet demand for the application feature it supports. This enables teams to right-size infrastructure needs, accurately measure the cost of a feature, and maintain availability if a service experiences a spike in demand.

- Easy Deployment

Microservices enable continuous integration and continuous delivery, making it easy to try out new ideas and to roll back if something doesn't work. The low cost of failure enables experimentation, makes it easier to update code, and accelerates time-to-market for new features.

Benefits of Microservices

- Technological Freedom

Microservices architectures don't follow a "one size fits all" approach. Teams have the freedom to choose the best tool to solve their specific problems. As a consequence, teams building microservices can choose the best tool for each job.

- Reusable Code

Dividing software into small, well-defined modules enables teams to use functions for multiple purposes. A service written for a certain function can be used as a building block for another feature. This allows an application to bootstrap off itself, as developers can create new capabilities without writing code from scratch.

- Resilience

Service independence increases an application's resistance to failure. In a monolithic architecture, if a single component fails, it can cause the entire application to fail. With microservices, applications handle total service failure by degrading functionality and not crashing the entire application.

Activity - Group Discussion

Activity - Group Discussion

There are lot of complexity when we are using or moving to microservices architecture. As a group, how to solve these challenges when we are moving to microservices

- Overcoming Design Complexity
- Achieving Data Consistency
- Need for Testing and Monitoring
- Debugging Issues
- Compromised Security
- Increased Operational Complexity
- Inter-Service Communication Breakdown
- Requires Team Expertise
- Maintenance of Microservices
- Network Management

Choose at least 5 challenges and break down the solution to how to solve them.

Activity

Learner:

- Clean up AWS.
- Remove/delete/terminate all service/ resources that you created.

Instructor

- Clean up AWS.
- Remove/delete/terminate all service/ resources that you created.
- Check the AWS account after learner clean up.

Submit Group Activity Docs/PPT as Assignment in Blackboard.

If still have time, few group can present their discussion.