



**NANYANG  
TECHNOLOGICAL  
UNIVERSITY**  
SINGAPORE

# Deploying Cloud Native App

Cloud Infrastructure Engineering

**Nanyang Technological University  
& Skills Union - 2022/2023**

# Course Content

- Self Study Check In
- Cloud Native Apps Concepts
- Key principles of cloud-native architecture
- 3 Benefits of cloud-native applications
- 4 examples of cloud-native applications
- Common Cloud Native Apps Knowledge
- Running your first Cloud Native application (Serverless)
  - Installation - Node, NPM, NVM

Q1: What is Cloud Native Application based on your knowledge?



Q2: It is easier to secure one monolithic application hosted on a single powerful server than compared to a distributed, cloud-native environment. True or False?  
Why?

Q3: Do you think that when we implement Containerization on our system, it make it complicated?



Q3: Do you think that when we migrate existing monolith system to a microservices architecture, it make it complicated?



# Activity

Instructor

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

# Activity

Take 5-10 Mins for Learners create new repository, then clone it to your own local computer.

Create a professional naming convention, so that it will look good on your github resume

REPO Name: first-node-application



# Cloud Native Applications



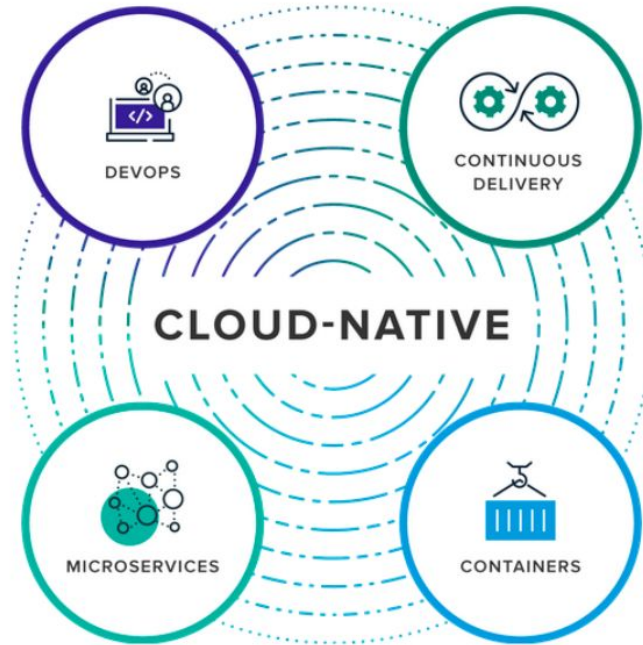
# What is Cloud Native Applications?

Cloud native applications—or native cloud applications (NCAs)—are programs designed for a cloud computing architecture.

Cloud-native applications are a collection of small, independent, and loosely coupled services. They are designed to deliver well-recognized business value, like the ability to rapidly incorporate user feedback for continuous improvement.

In short, cloud-native app development is a way to speed up how you build new applications, optimize existing ones, and connect them all. Its goal is to deliver apps users want at the pace a business needs.

# Key Principles of Cloud Native Architecture



# Key Principles of Cloud Native Architecture - Microservices

**Microservices** are considered an architectural strategy capable of managing complex applications simply. They are highly capable of communicating with others which is possible with a API since it is an **independent process**. Each service is confined and focuses on completing the scheduled tasks on time.

Microservice architectures are well known for their modern application-building strategy that allows you to reuse and redesign specific areas based on your requirements. Enterprises can adopt microservices for easy deployment, scaling, and development and support technological diversity.



# Key Principles of Cloud Native Architecture - Containerization

Containers are now mainstream to developers and enterprises for building cloud-native applications. Linux namespaces are used to handle the space between network stacks, processes, and file systems. Containers are highly secured partitions that perform best by following the namespace method. Each container runs one or more multiple Linux processes and is supported by Linux Kernel on the host.

It works like virtual machines (VMs) do and is highly flexible for running operations smoothly. **VMs are only installed with the help of a full operating system, whereas containers support apps using the packaging software.** Developers and enterprises efficiently manage their applications with a packaging approach. Containers have less weight than VMs and require less maintenance and fewer resources to process the cloud-native apps.

Most enterprises prefer containers for faster accessibility, higher portability, and easy deployment.

# Key Principles of Cloud Native Architecture - CI/CD

Continuous integration (CI) assists in automating the integration of new code in a repository so that multiple contributors can have access to working on the same project simultaneously. CI tools are capable enough to run automated tests and verify that new bugs are not present in the code. It also prohibits overwriting other developers' work and automatically creates new builds once the verification process is completed.

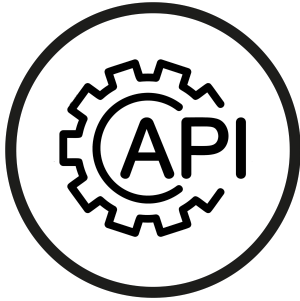
In addition, it uses these methodologies in cloud-native software development. Cloud-native CI tools and processes allow a team's DevOps experts to work together on a similar codebase and update the software frequently without compromising the quality. Continuous delivery (CD) helps deploy new code builds for production and testing processes. Besides all that, the software release cycle is streamlined.

# Key Principles of Cloud Native Architecture - DevOps

The cloud-native approach is DevOps' best practice to automate the process between operations and software development teams. It is bringing innovation to the global market at a rapid pace and helping enterprises to transform along with DevOps to align the processes and technologies for boosting business efficiency and productivity.

The collaboration of DevOps with the CI/CD paradigm has benefitted developers in iterating software to improve deployment time and reduce errors. From an enterprise perspective, a cloud-native approach with DevOps for software development helps maximize agility via continuous deployment and allows apps to scale up without needing further changes.

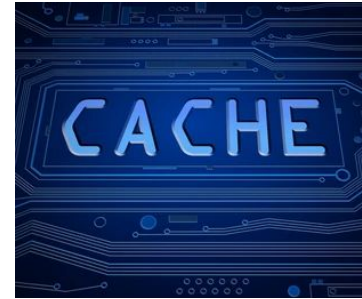
# Common Cloud Native Apps Knowledge



API



Logging



Cache



# API

Q: What other API technologies are available besides the RESTful API to your knowledge?



# API

## gRPC vs. REST vs. GraphQL comparison

Feature	GraphQL	REST	gRPC
Data fetch	Efficient fetching of data. Only required data will be fetched from server.	Extra data might be returned by the server unless new endpoints/query filters are defined on server side.	Extra data might be returned by server unless new endpoints/filters are defined on server side.
HTTP 1.1 vs HTTP 2	Follows request-response model. It can work with either HTTP version but is typically built with HTTP 1.1.	Follows request-response model. It can work with either HTTP version but is still typically built with HTTP 1.1.	Follows client-response model and is based on HTTP 2. Some servers have workarounds to make it work with HTTP 1.1 but it is not the default.
Browser support	Works everywhere.	Works everywhere.	Limited support. Need to use gRPC-Web, which is an extension of gRPC for the web and is based on HTTP 1.1.
Payload data structure	Uses JSON-based payloads to send/receive data.	Mostly uses JSON- and XML-based payloads to send/receive data.	Uses Protocol Buffers by default to serialize payload data.
Code generation	Need to use third-party tools like GraphQL Code Generator to generate client code. Docs and an interactive playground can be natively generated by using GraphiQL.	Need to use third-party tools like Swagger to generate client code.	gRPC has native support for code generation for various target languages.
Request caching	Hard to cache requests by default as every request is sent to the same endpoint.	Easy to cache requests on the client and server sides. Most clients/servers natively support it.	Doesn't support request/response caching by default.

# Logging

Q: What service on AWS that will handle logging for the services?

# Logging

A microservices-based application might have several services running and communicating amongst themselves.

Things get complicated when one or more services fail, and you need to know which one failed and why.

It's also essential that you're able to comprehend the whole request flow — which services were invoked, how many times, and in what order.

# Caching

Q: What service that available on AWS related to caching?

<https://aws.amazon.com/caching/aws-caching/>



# Caching

Caching improves availability, scalability, and performance of Microservices by reducing round-trips to dependencies.

In computing, a cache is a high-speed data storage layer which stores a subset of data, typically transient in nature, so that future requests for that data are served up faster than is possible by accessing the data's primary storage location. Caching allows you to efficiently reuse previously retrieved or computed data.

Querying database every time is nonessential. Perhaps we can save our applications a little compute by serving data from the buffer. This is profitable when the nature of data is static, and don't change very often.

# Caching

Below are the 4 things to consider when deciding to use the cache.

- What to cache.
- When to cache.
- Where to cache.
- How long to cache.

Break 10 Mins





# Activity

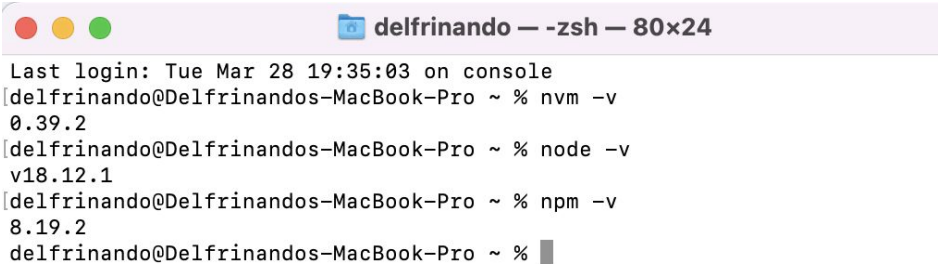
Install these on your local machine:

- <https://nodejs.org/en/download/>
- <https://docs.npmjs.com/downloading-and-installing-node-js-and-npm>
- <https://github.com/nvm-sh/nvm>

Check on your terminal using command:

`node -v`

`npm -v`

A screenshot of a terminal window titled "delfrinando — zsh — 80x24". The terminal shows the output of running "nvm -v" (0.39.2), "node -v" (v18.12.1), and "npm -v" (8.19.2). The prompt is "delfrinando@Delfrinandos-MacBook-Pro ~ %".

```
Last login: Tue Mar 28 19:35:03 on console
[delfrinando@Delfrinandos-MacBook-Pro ~ % nvm -v
0.39.2
[delfrinando@Delfrinandos-MacBook-Pro ~ % node -v
v18.12.1
[delfrinando@Delfrinandos-MacBook-Pro ~ % npm -v
8.19.2
delfrinando@Delfrinandos-MacBook-Pro ~ % █
```

# Activity

Instructor demonstrate on how to create your first node application.

```
$ npm init
```

Instructor explain about the project structure including “package.json” and “node\_modules”.

# Activity

Add this to index.js

<https://github.com/su-ntu-ctp/6m-cloud-3.3-deploying-cloud-native-app/blob/main/simple-node-application/index.js>

Run project using “node index.js”

# Activity

Add this to index.js

```
const http = require("http");

const hostname = "localhost";

const port = 3700;

const server = http.createServer((req, res) => {

  console.log(req.headers);

  res.statusCode = 200;

  res.end("<html><body><h1>Hello, World!</h1></body></html>");

})

server.listen(port, hostname);
```

Run project using "node index.js"

# Activity

Take 5-10 Mins for Learners push the changes to the repository

# 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.

# What's Next?

- Improve your personal knowledge about git command. All of the command for git will definitely be used on the real project
- Install docker on your computer before the next lesson.  
<https://docs.docker.com/get-docker/>