Instructor Notes:

Add instructor notes here.

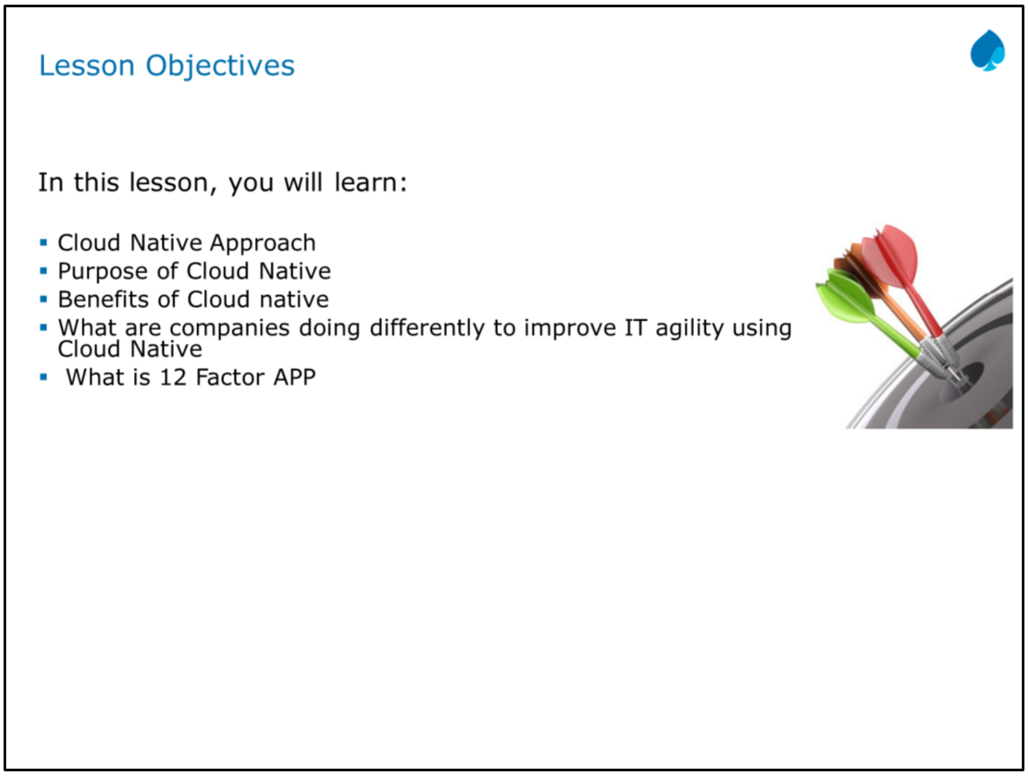
Cloud Native Concepts

Lesson 02: Cloud Native Concepts

Capgemini

Instructor Notes:

This lesson is to give an Introduction on Java Server Pages



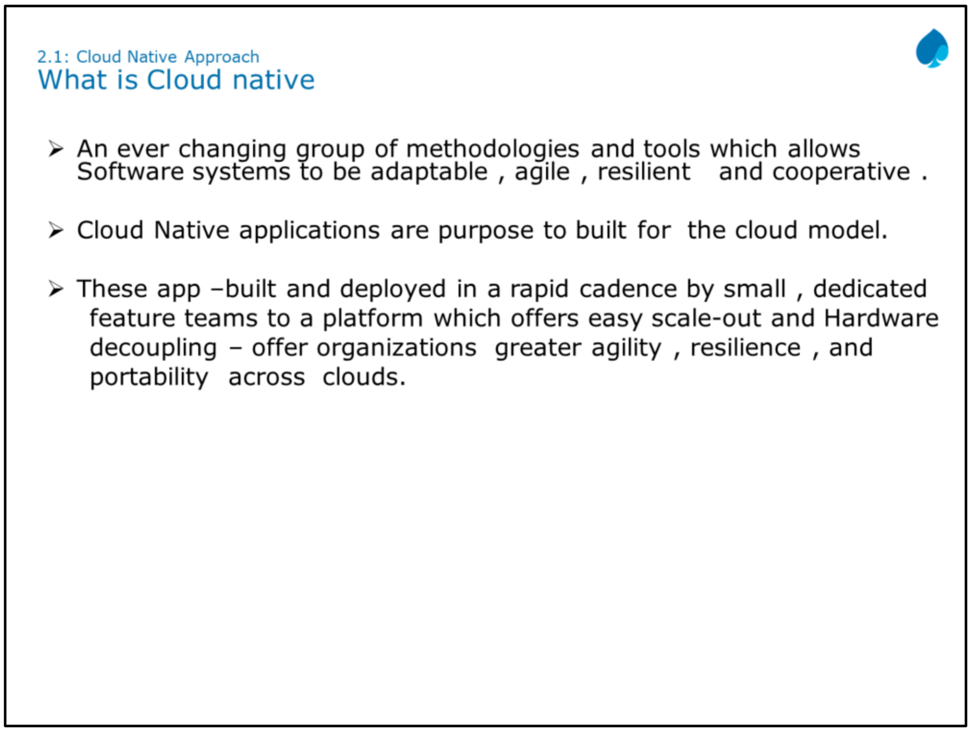
Lesson Objectives

In this lesson, you will learn:

- Cloud Native Approach
- Purpose of Cloud Native
- Benefits of Cloud native
- What are companies doing differently to improve IT agility using Cloud Native
- What is 12 Factor APP

Cloud Native Approach
Purpose of Cloud Native
What are Cloud Native companies doing differently to improve IT agility
Benefits of Cloud native
What are Cloud Native companies doing differently to improve IT agility

Instructor Notes:



2.1: Cloud Native Approach

What is Cloud native

- An ever changing group of methodologies and tools which allows Software systems to be adaptable , agile , resilient and cooperative .
- Cloud Native applications are purpose to built for the cloud model.
- These app –built and deployed in a rapid cadence by small , dedicated feature teams to a platform which offers easy scale-out and Hardware decoupling – offer organizations greater agility , resilience , and portability across clouds.

The world is changing first :

- Since the year 2000, 56 % of the Fortune 500 companies are no longer in the list.
- Disruption is everywhere – May be System issue , network issue
- Learn to Adapt the new concept- Technology are changing

Demand of Modern World application :

- Adaptability and Agility :- The system should be able to respond quickly to change.(Hardware / SW , business model etc.)
- Resilience :- A system should be able to recover from failure (i.e. SW issue , Physical failure, attacks etc.)
- Cooperation :- A system should be able to Work together with other systems, means the complex systems are made of simpler systems working together.

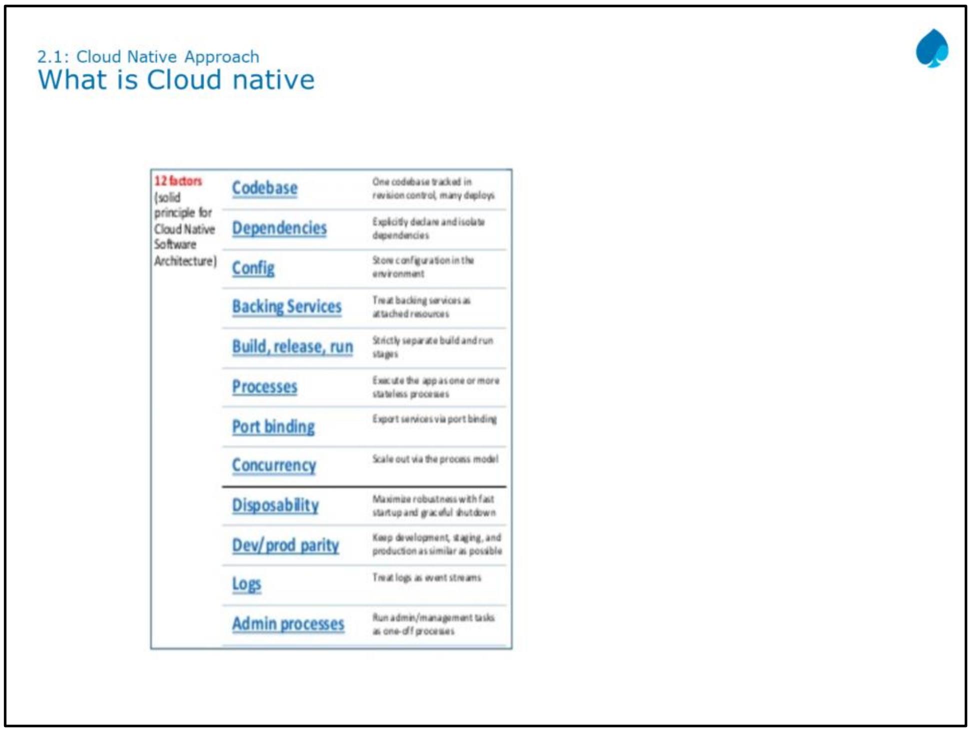
The Cloud Native toolbox Automation Infrastructure as code Platform as a Service Containerization 12 factor apps Continuous Integration & Delivery Source control Multi Data Center

Infrastructure :-- Platform as a Service A way to develop, run and manage applications without going into the complexities of managing the underlying infrastructure. Heroku Amazon Elastic Beanstalk Cloud Foundry (IBM Bluemix, Pivotal Web Services, GE Predix) Google App Engine OpenShift
Infrastructure Automation :--as code A way to manage and provision infrastructure from the ground up with a high level or descriptive language. Terraform Puppet Chef Ansible BOSH

Runtime Platform :-- Containerization A technique of encapsulating applications in containers with its own operating environment, abstracting them from the operating system. LXC Docker Rocket Warden/Garden (Cloud Foundry)

12 factor apps A methodology of writing applications that allows system and software architects, and developers, to create software that can run in the cloud seamlessly.
1. Codebase in revision
2. Declare dependencies
3. Config in environment
4. Backing services = attached resources
5. Separate build and run stages
6. Export services via port binding
7. Scale out via process model
8. Fast startup, graceful shutdown
9. Environments parity
10. Logs = events streams
11. Admin processes as tasks

Instructor Notes:


2.1: Cloud Native Approach

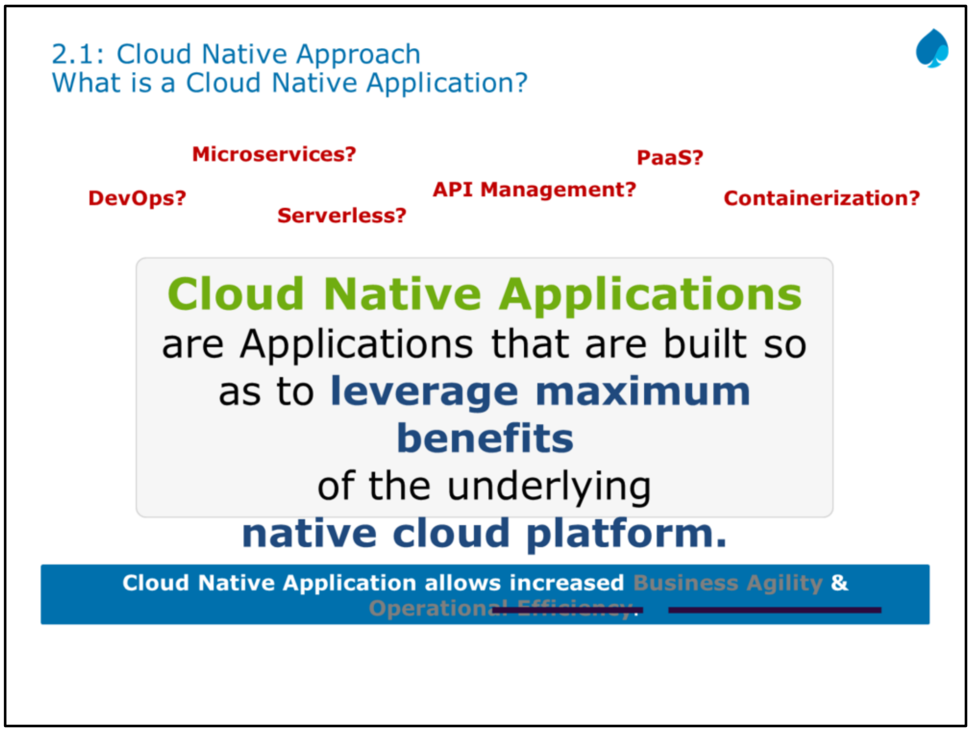
What is Cloud native

12 factors (solid principle for Cloud Native Software Architecture)	Codebase	One codebase tracked in revision control, many deploys
	Dependencies	Explicitly declare and isolate dependencies
	Config	Store configuration in the environment
	Backing Services	Treat backing services as attached resources
	Build, release, run	Strictly separate build and run stages
	Processes	Execute the app as one or more stateless processes
	Port binding	Export services via port binding
	Concurrency	Scale out via the process model
	Disposability	Maximize robustness with fast startup and graceful shutdown
	Dev/prod parity	Keep development, staging, and production as similar as possible
	Logs	Treat logs as event streams
	Admin processes	Run admin/management tasks as one-off processes

12 factor apps A methodology of writing applications that allows system and software architects, and developers, to create software that can run in the cloud seamlessly.

1. Codebase in revision
2. Declare dependencies
3. Config in environment
4. Backing services = attached resources
5. Separate build and run stages
6. Export services via port binding
7. Scale out via process model
8. Fast startup, graceful shutdown
9. Environments parity
10. Logs = events streams
11. Admin processes as tasks

Instructor Notes:



2.1: Cloud Native Approach
What is a Cloud Native Application?

Microservices? PaaS? DevOps? API Management? Containerization? Serverless?

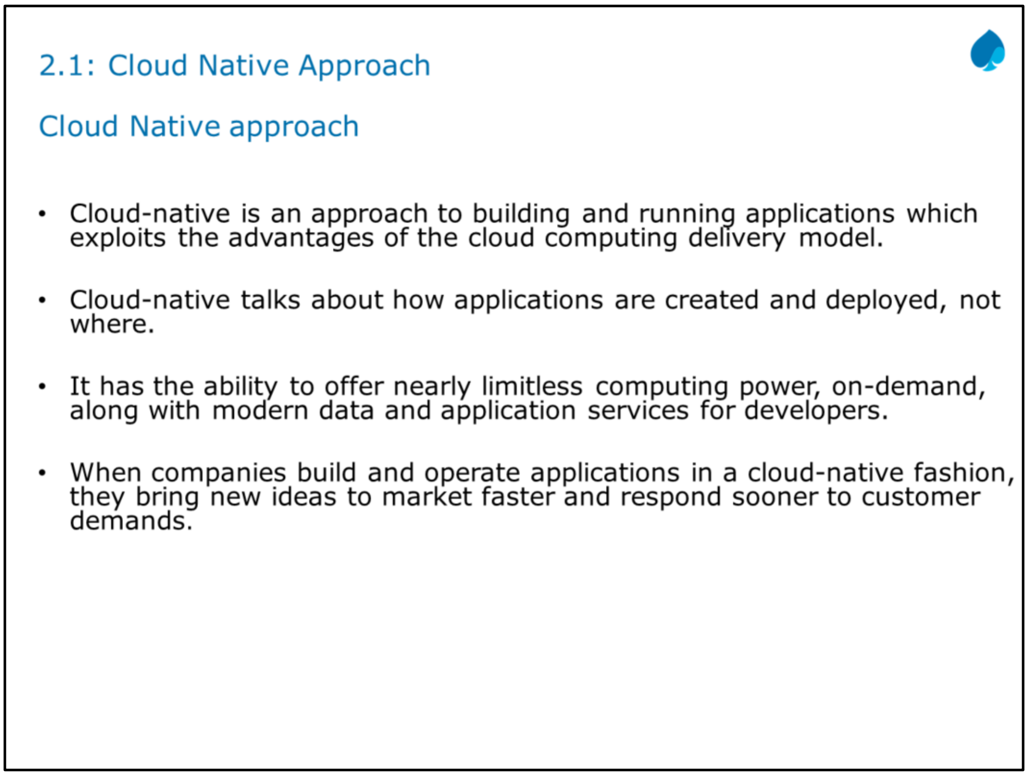
Cloud Native Applications are Applications that are built so as to **leverage maximum benefits** of the underlying **native cloud platform.**

Cloud Native Application allows increased Business Agility & Operational Efficiency.

Cloud Native Applications
are Applications that are built so as to **leverage maximum benefits**
of the underlying
native cloud platform

Instructor Notes:

Explain role of JSP in web applications and “big picture”



2.1: Cloud Native Approach

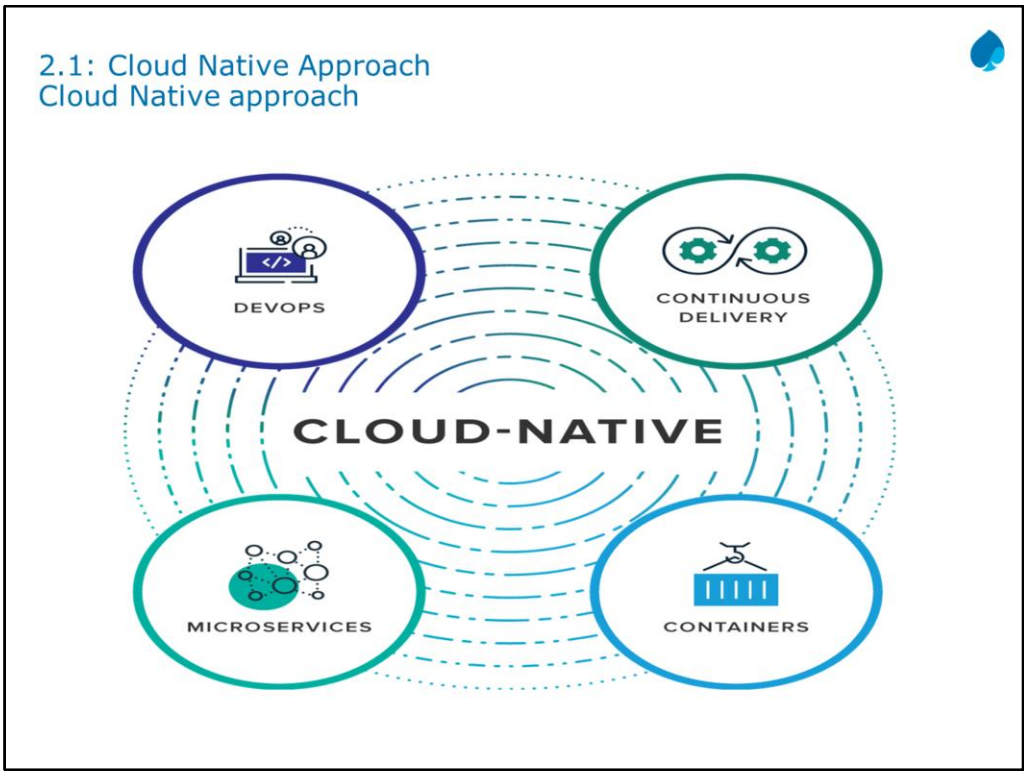
Cloud Native approach

- Cloud-native is an approach to building and running applications which exploits the advantages of the cloud computing delivery model.
- Cloud-native talks about how applications are created and deployed, not where.
- It has the ability to offer nearly limitless computing power, on-demand, along with modern data and application services for developers.
- When companies build and operate applications in a cloud-native fashion, they bring new ideas to market faster and respond sooner to customer demands.

Cloud-native is an approach to building and running applications which exploits the advantages of the cloud computing delivery model.
Cloud-native talks about how applications are created and deployed, not where.
Most important is the ability to offer nearly limitless computing power, on-demand, along with modern data and application services for developers.
When companies build and operate applications in a cloud-native fashion, they bring new ideas to market faster and respond sooner to customer demands.

Instructor Notes:

Explain role of JSP in web applications and “big picture”



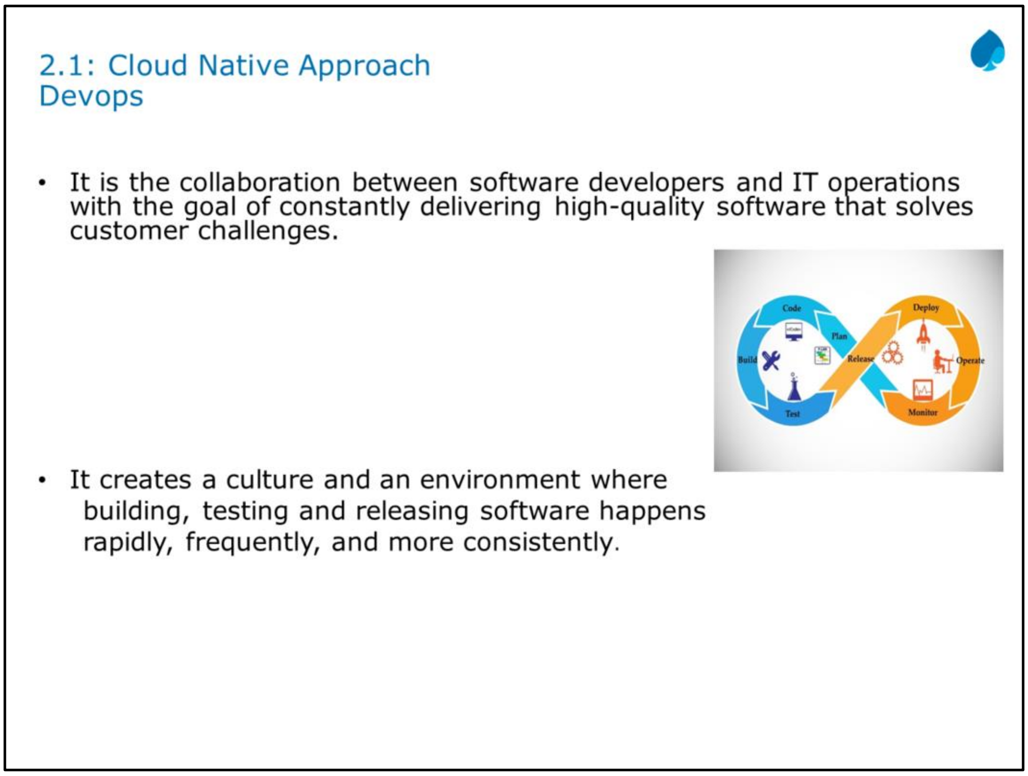
2.1: Cloud Native Approach
Cloud Native approach

Organizations require a platform for building and operating cloud-native applications and services that automates and integrates the concepts of

- DevOps,
- Continuous delivery,
- Microservices and
- Containers:

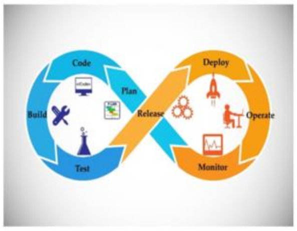
Instructor Notes:

Explain role of JSP in web applications and “big picture”



2.1: Cloud Native Approach
Devops

- It is the collaboration between software developers and IT operations with the goal of constantly delivering high-quality software that solves customer challenges.

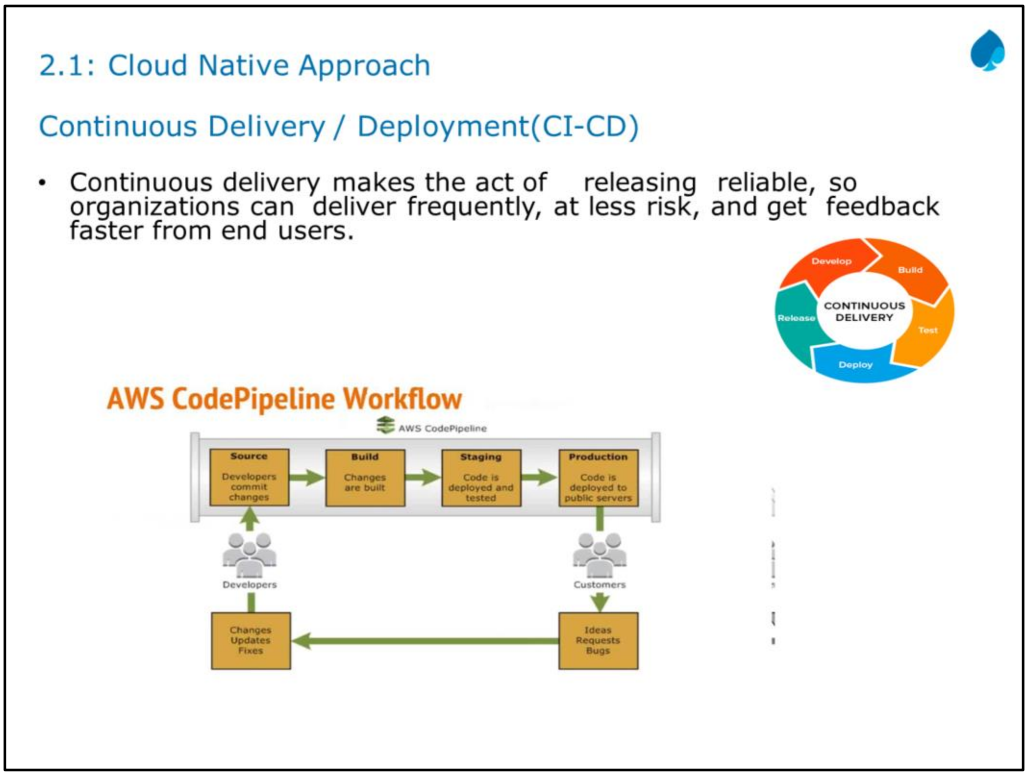


- It creates a culture and an environment where building, testing and releasing software happens rapidly, frequently, and more consistently.

is the collaboration between software developers and IT operations with the goal of constantly delivering high-quality software that solves customer challenges. It has the potential to create a culture and an environment where building, testing and releasing software happens rapidly, frequently, and more consistently. Continuous Delivery, enabled by Agile product development practices, is about shipping small batches of software to production constantly, through automation. Continuous delivery makes the act of releasing dull and reliable, so organizations can deliver frequently, at less risk, and get feedback faster from end users.

Instructor Notes:

Explain role of JSP in web applications and “big picture”



2.1: Cloud Native Approach

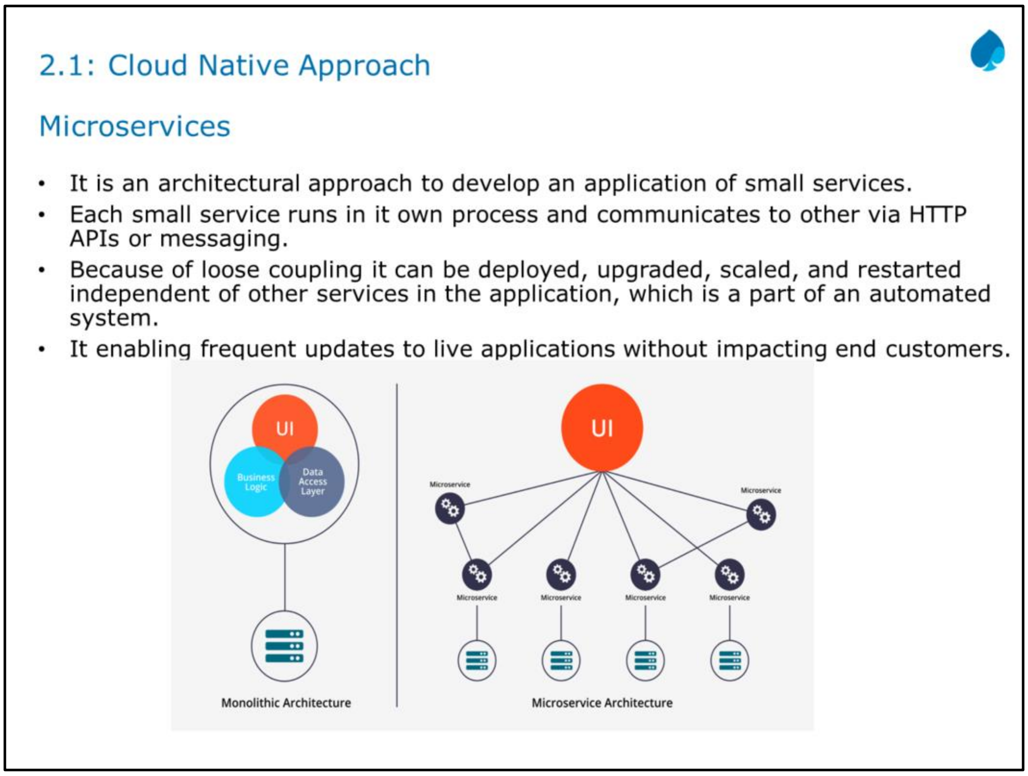
Continuous Delivery / Deployment(CI-CD)

- Continuous delivery makes the act of releasing reliable, so organizations can deliver frequently, at less risk, and get feedback faster from end users.

is the collaboration between software developers and IT operations with the goal of constantly delivering high-quality software that solves customer challenges. It has the potential to create a culture and an environment where building, testing and releasing software happens rapidly, frequently, and more consistently. Continuous Delivery, enabled by Agile product development practices, is about shipping small batches of software to production constantly, through automation. Continuous delivery makes the act of releasing dull and reliable, so organizations can deliver frequently, at less risk, and get feedback faster from end users.

Instructor Notes:

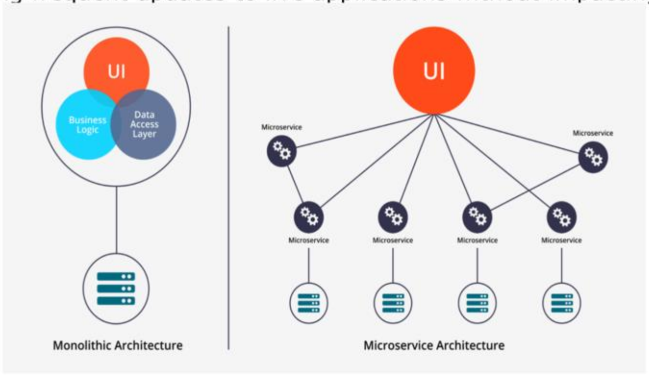
Explain role of JSP in web applications and “big picture”



2.1: Cloud Native Approach

Microservices

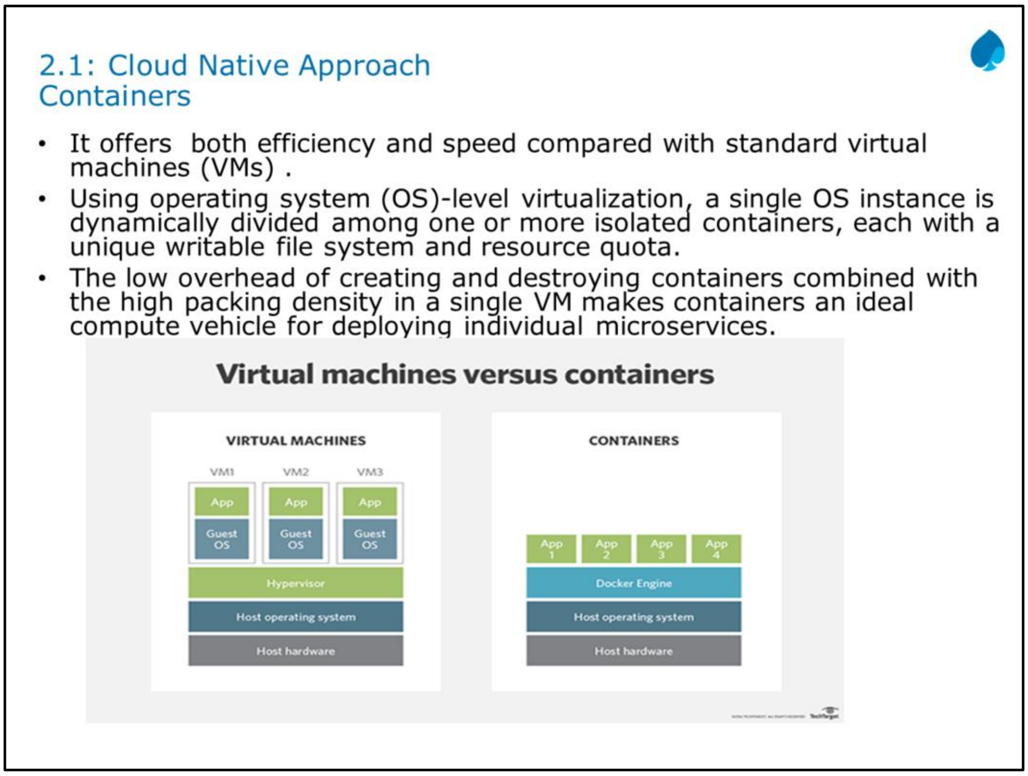
- It is an architectural approach to develop an application of small services.
- Each small service runs in it own process and communicates to other via HTTP APIs or messaging.
- Because of loose coupling it can be deployed, upgraded, scaled, and restarted independent of other services in the application, which is a part of an automated system.
- It enabling frequent updates to live applications without impacting end customers.



Microservices is an architectural approach to developing an application as a collection of small services; each service implements business capabilities, runs in its own process and communicates via HTTP APIs or messaging. Each microservice can be deployed, upgraded, scaled, and restarted independent of other services in the application, typically as part of an automated system, enabling frequent updates to live applications without impacting end customers

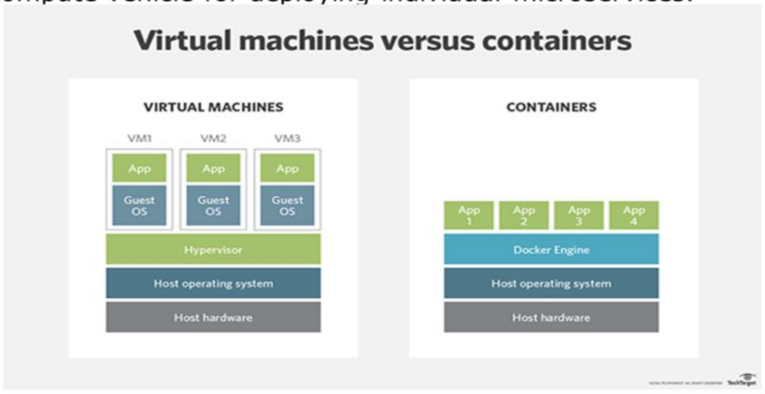
Instructor Notes:

Explain role of JSP in web applications and “big picture”



2.1: Cloud Native Approach Containers

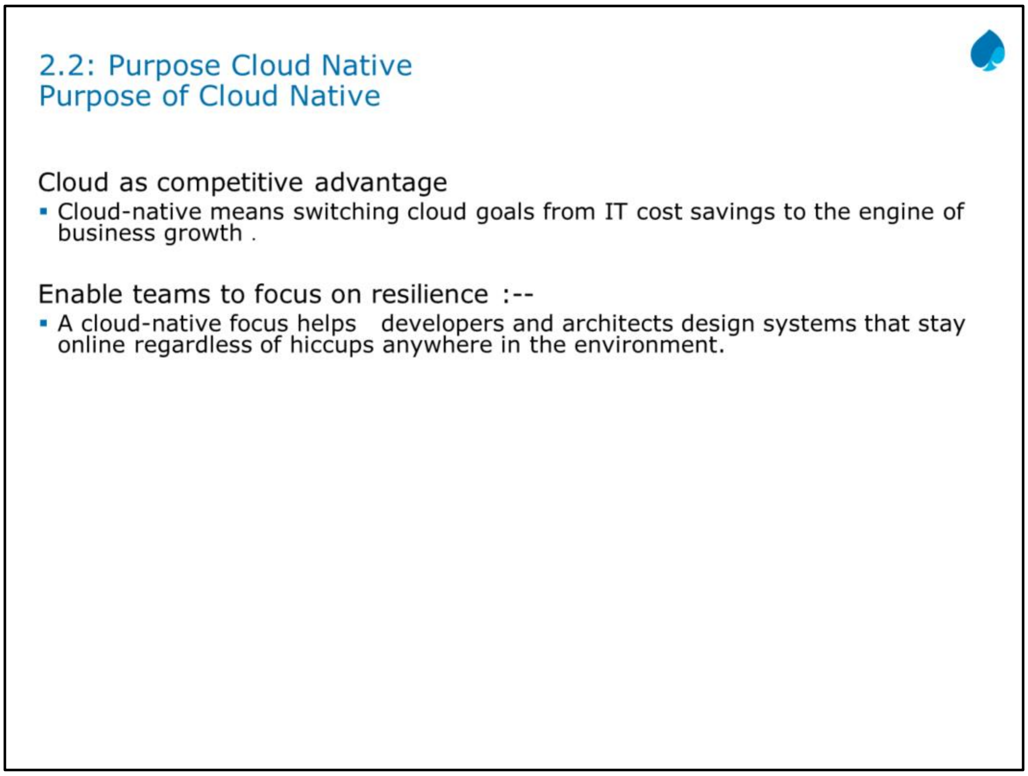
- It offers both efficiency and speed compared with standard virtual machines (VMs) .
- Using operating system (OS)-level virtualization, a single OS instance is dynamically divided among one or more isolated containers, each with a unique writable file system and resource quota.
- The low overhead of creating and destroying containers combined with the high packing density in a single VM makes containers an ideal compute vehicle for deploying individual microservices.



Containers offer both efficiency and speed compared with standard virtual machines (VMs). Using operating system (OS)-level virtualization, a single OS instance is dynamically divided among one or more isolated containers, each with a unique writable file system and resource quota. The low overhead of creating and destroying containers combined with the high packing density in a single VM makes containers an ideal compute vehicle for deploying individual microservices.

Instructor Notes:

Explain role of JSP in web applications and “big picture”



2.2: Purpose Cloud Native
Purpose of Cloud Native

Cloud as competitive advantage

- Cloud-native means switching cloud goals from IT cost savings to the engine of business growth .

Enable teams to focus on resilience :--

- A cloud-native focus helps developers and architects design systems that stay online regardless of hiccups anywhere in the environment.

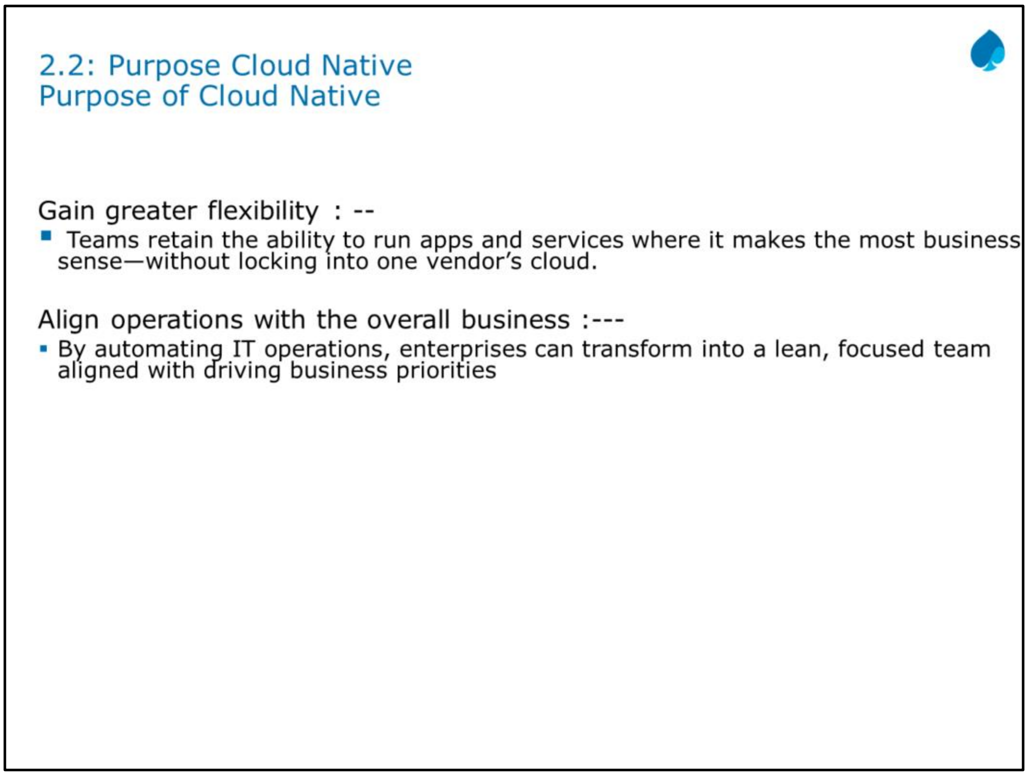
Cloud as competitive advantage :---

Cloud-native means switching cloud goals from IT cost savings to the engine of business growth. In the age of software, businesses that can quickly build and deliver applications in response to customer needs will build enduring success.

Enable teams to focus on resilience.
When legacy infrastructure fails, services can suffer. In a cloud-native world, teams focus specifically on architecting for resilience. A cloud-native focus helps developers and architects design systems that stay online regardless of hiccups anywhere in the environment.

Instructor Notes:

Explain role of JSP in web applications and "big picture"



2.2: Purpose Cloud Native
Purpose of Cloud Native

Gain greater flexibility : --

- Teams retain the ability to run apps and services where it makes the most business sense—without locking into one vendor's cloud.

Align operations with the overall business :---

- By automating IT operations, enterprises can transform into a lean, focused team aligned with driving business priorities

Gain greater flexibility:-

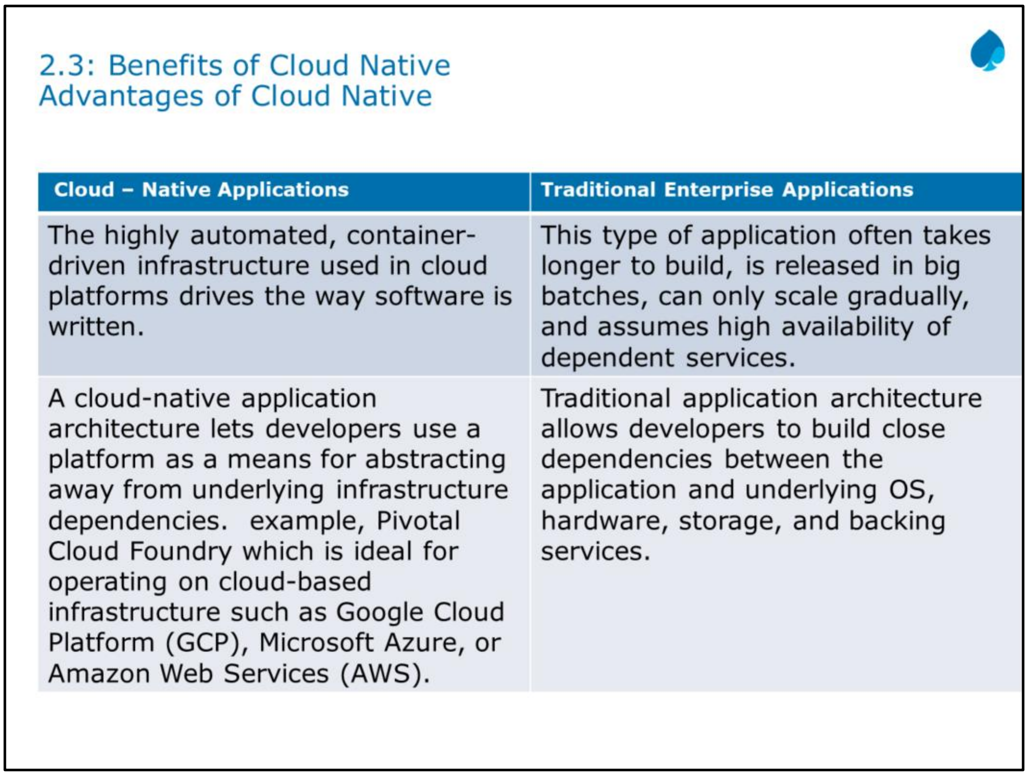
Public cloud providers continue to offer impressive services at reasonable costs. But most enterprises aren't ready to choose just one infrastructure. With a platform that supports a cloud-native approach, enterprises build applications that run on any (public or private) cloud without modification. Teams retain the ability to run apps and services where it makes the most business sense—without locking into one vendor's cloud.

Align operations with the overall business :--

By automating IT operations, enterprises can transform into a lean, focused team aligned with driving business priorities. They eliminate the risk of failure due to human error as staff focus on automated improvements to replace routine, mundane admin tasks. With automated live patching and upgrades at all levels of the stack, they eliminate downtime and the need for ops experts with 'hand-me-down' expertise.

Instructor Notes:

Explain role of JSP in web applications and “big picture”

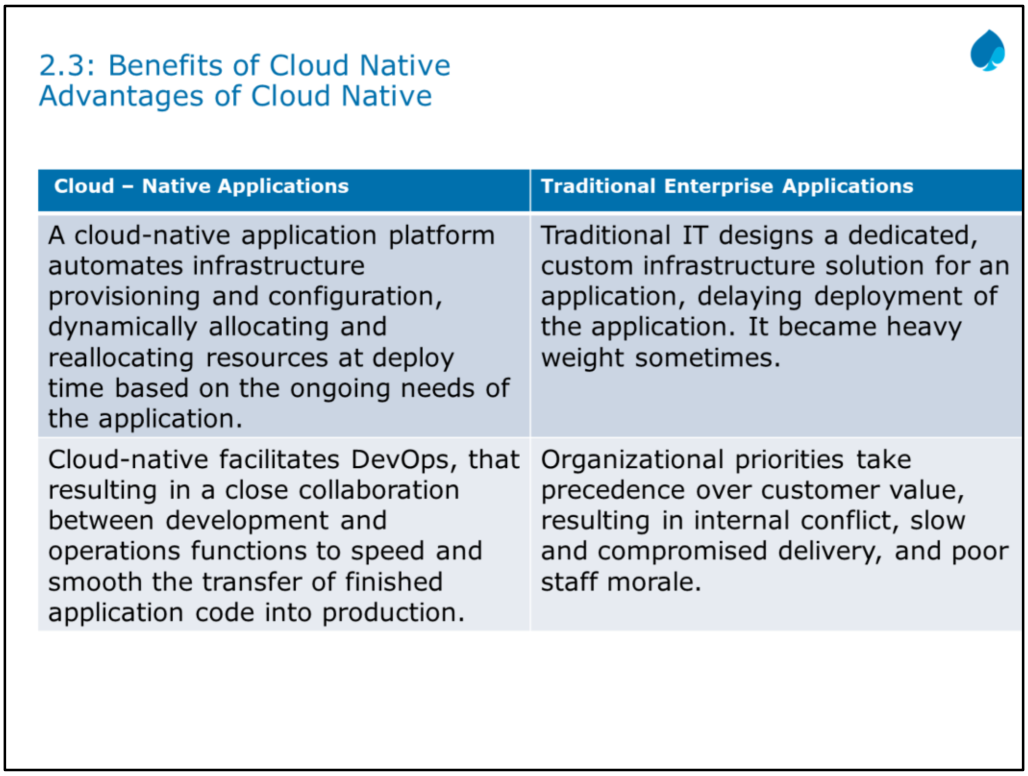


2.3: Benefits of Cloud Native
Advantages of Cloud Native

Cloud – Native Applications	Traditional Enterprise Applications
The highly automated, container-driven infrastructure used in cloud platforms drives the way software is written.	This type of application often takes longer to build, is released in big batches, can only scale gradually, and assumes high availability of dependent services.
A cloud-native application architecture lets developers use a platform as a means for abstracting away from underlying infrastructure dependencies. example, Pivotal Cloud Foundry which is ideal for operating on cloud-based infrastructure such as Google Cloud Platform (GCP), Microsoft Azure, or Amazon Web Services (AWS).	Traditional application architecture allows developers to build close dependencies between the application and underlying OS, hardware, storage, and backing services.

Instructor Notes:

Explain role of JSP in web applications and “big picture”



2.3: Benefits of Cloud Native
Advantages of Cloud Native

Cloud – Native Applications	Traditional Enterprise Applications
A cloud-native application platform automates infrastructure provisioning and configuration, dynamically allocating and reallocating resources at deploy time based on the ongoing needs of the application.	Traditional IT designs a dedicated, custom infrastructure solution for an application, delaying deployment of the application. It became heavy weight sometimes.
Cloud-native facilitates DevOps, that resulting in a close collaboration between development and operations functions to speed and smooth the transfer of finished application code into production.	Organizational priorities take precedence over customer value, resulting in internal conflict, slow and compromised delivery, and poor staff morale.

Gain greater flexibility:-

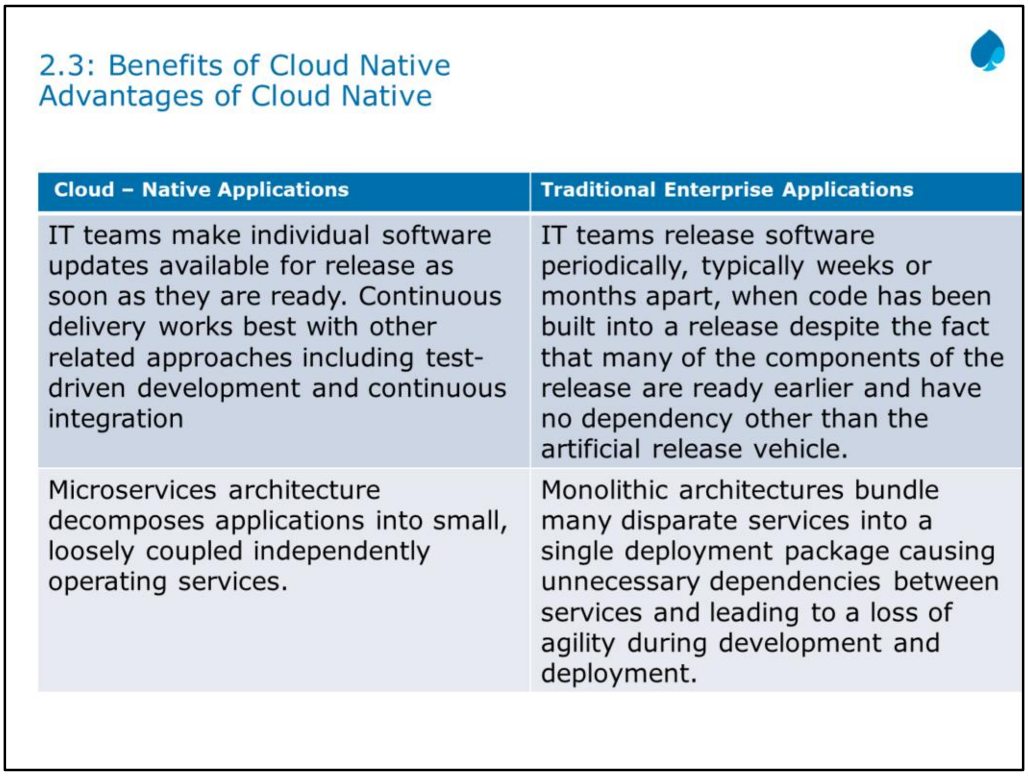
Public cloud providers continue to offer impressive services at reasonable costs. But most enterprises aren’t ready to choose just one infrastructure. With a platform that supports a cloud-native approach, enterprises build applications that run on any (public or private) cloud without modification. Teams retain the ability to run apps and services where it makes the most business sense—without locking into one vendor’s cloud.

Align operations with the overall business :--

By automating IT operations, enterprises can transform into a lean, focused team aligned with driving business priorities. They eliminate the risk of failure due to human error as staff focus on automated improvements to replace routine, mundane admin tasks. With automated live patching and upgrades at all levels of the stack, they eliminate downtime and the need for ops experts with ‘hand-me-down’ expertise.

Instructor Notes:

Explain role of JSP in web applications and “big picture”

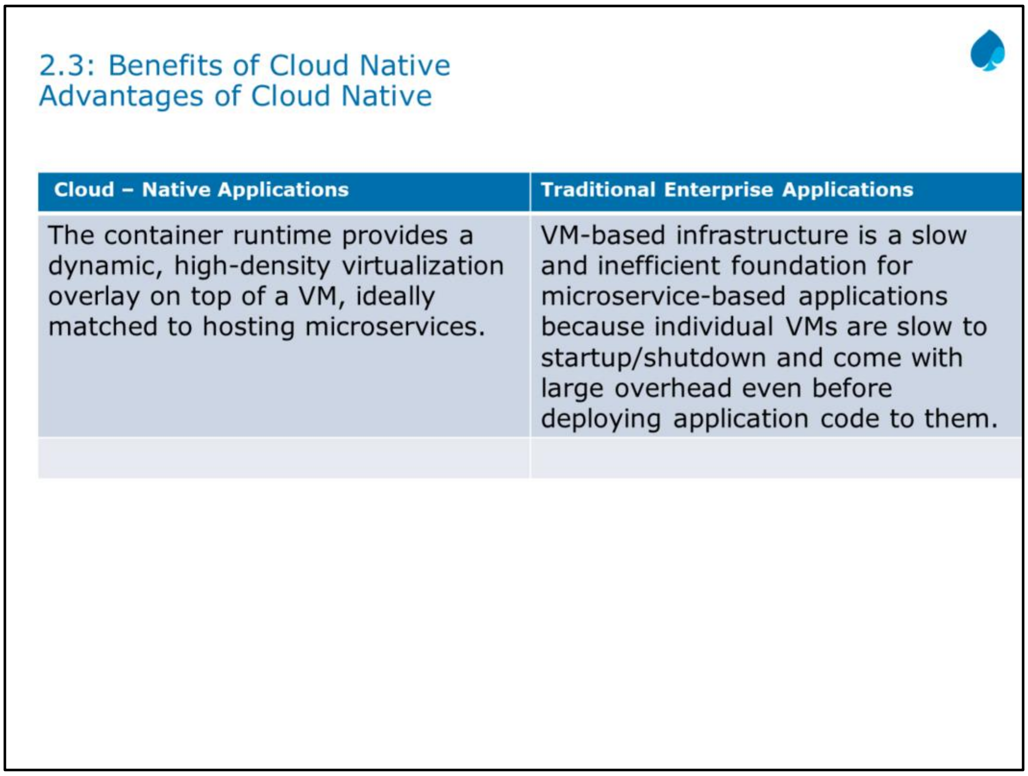


2.3: Benefits of Cloud Native
Advantages of Cloud Native

Cloud – Native Applications	Traditional Enterprise Applications
IT teams make individual software updates available for release as soon as they are ready. Continuous delivery works best with other related approaches including test-driven development and continuous integration	IT teams release software periodically, typically weeks or months apart, when code has been built into a release despite the fact that many of the components of the release are ready earlier and have no dependency other than the artificial release vehicle.
Microservices architecture decomposes applications into small, loosely coupled independently operating services.	Monolithic architectures bundle many disparate services into a single deployment package causing unnecessary dependencies between services and leading to a loss of agility during development and deployment.

Instructor Notes:

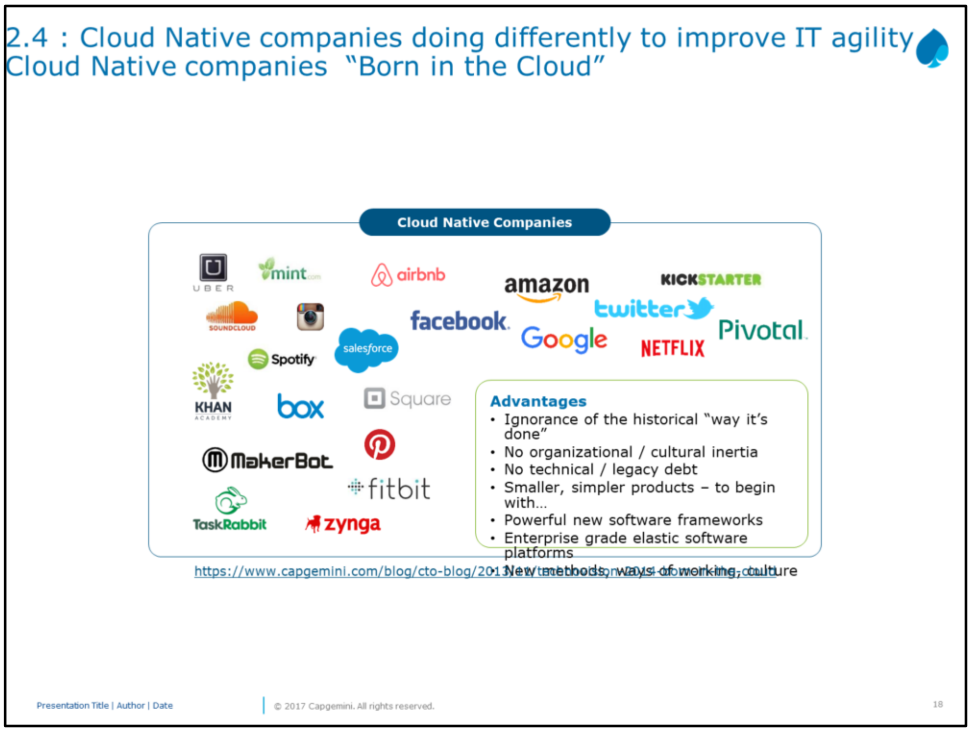
Explain role of JSP in web applications and “big picture”



2.3: Benefits of Cloud Native
Advantages of Cloud Native

Cloud – Native Applications	Traditional Enterprise Applications
The container runtime provides a dynamic, high-density virtualization overlay on top of a VM, ideally matched to hosting microservices.	VM-based infrastructure is a slow and inefficient foundation for microservice-based applications because individual VMs are slow to startup/shutdown and come with large overhead even before deploying application code to them.

Instructor Notes:



2.4 : Cloud Native companies doing differently to improve IT agility
Cloud Native companies "Born in the Cloud"

Cloud Native Companies

UBER
mint
airbnb
amazon
KICKSTARTER
SOUNDCLOUD
facebook
twitter
Google
Pivotal
NETFLIX
salesforce
Spotify
KHAN ACADEMY
box
Square
MakerBot
fitbit
TaskRabbit
zynga

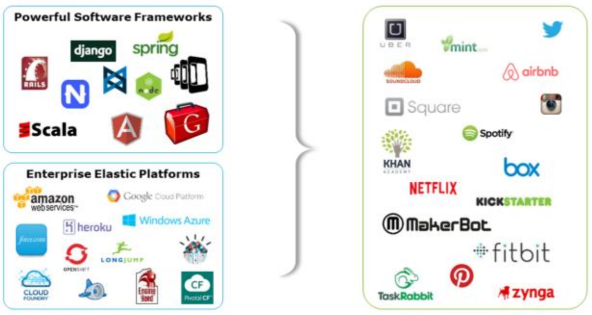
Advantages
- Ignorance of the historical "way it's done"
- No organizational / cultural inertia
- No technical / legacy debt
- Smaller, simpler products – to begin with…
- Powerful new software frameworks
- Enterprise grade elastic software platforms
- New methods, ways of working, culture

https://www.capgemini.com/blog/cto-blog/2013/[illegible]

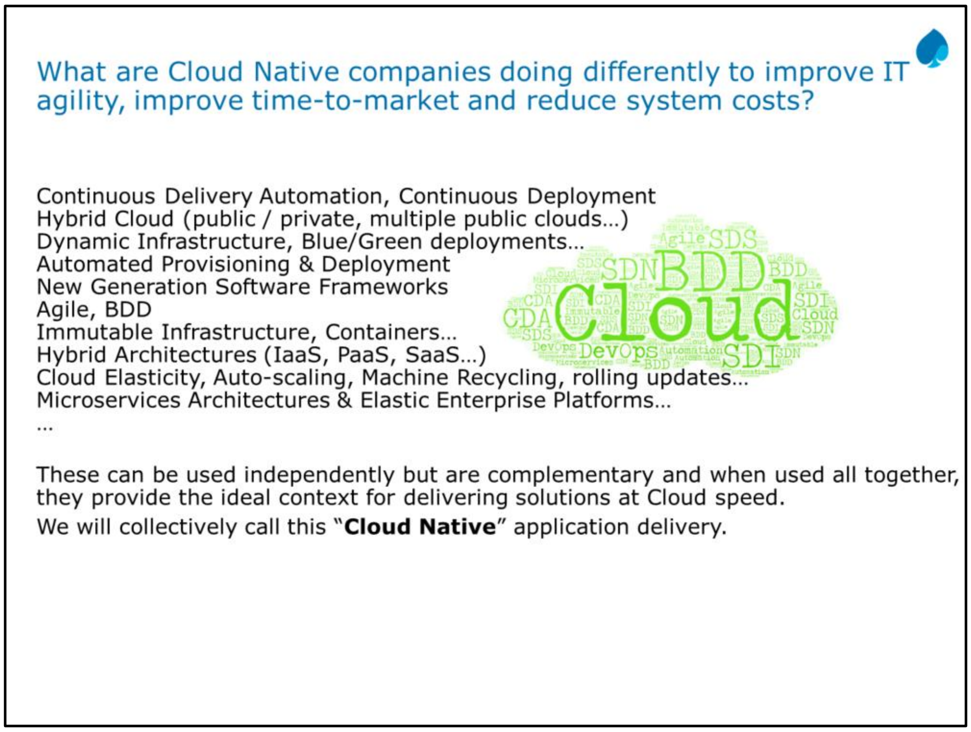
Presentation Title | Author | Date © 2017 Capgemini. All rights reserved. 18

Instructor Notes:

2.4 : Cloud Native companies doing differently to improve IT agility Different Frameworks and Platforms from typical enterprises…



Instructor Notes:



What are Cloud Native companies doing differently to improve IT agility, improve time-to-market and reduce system costs?

Continuous Delivery Automation, Continuous Deployment
Hybrid Cloud (public / private, multiple public clouds...)
Dynamic Infrastructure, Blue/Green deployments...
Automated Provisioning & Deployment
New Generation Software Frameworks
Agile, BDD
Immutable Infrastructure, Containers...
Hybrid Architectures (IaaS, PaaS, SaaS...)
Cloud Elasticity, Auto-scaling, Machine Recycling, rolling updates...
Microservices Architectures & Elastic Enterprise Platforms...
...

These can be used independently but are complementary and when used all together, they provide the ideal context for delivering solutions at Cloud speed.

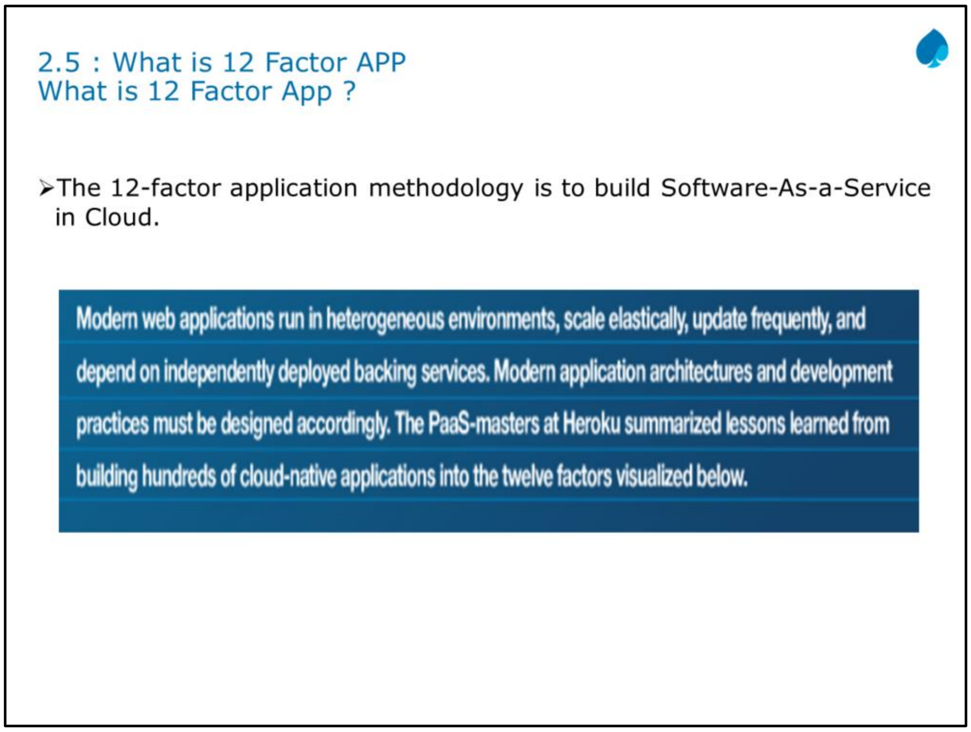
We will collectively call this "**Cloud Native**" application delivery.

Continuous Delivery Automation, Continuous Deployment
Hybrid Cloud (public / private, multiple public clouds…)
Software Defined Everything – SDI, SDN, SDS…
Dynamic Infrastructure, Blue/Green deployments…
Automated Provisioning & Deployment
New Generation Software Frameworks
Agile, BDD
Immutable Infrastructure, Containers…
Hybrid Architectures (IaaS, PaaS, SaaS…)
Cloud Elasticity, Auto-scaling, Machine Recycling, rolling updates…
Microservices Architectures & Elastic Enterprise Platforms…
…

These can be used independently but are complementary and when used all together, they provide the ideal context for delivering solutions at Cloud speed.

We will collectively call this "**Cloud Native**" application delivery.

Instructor Notes:



2.5 : What is 12 Factor APP
What is 12 Factor App ?

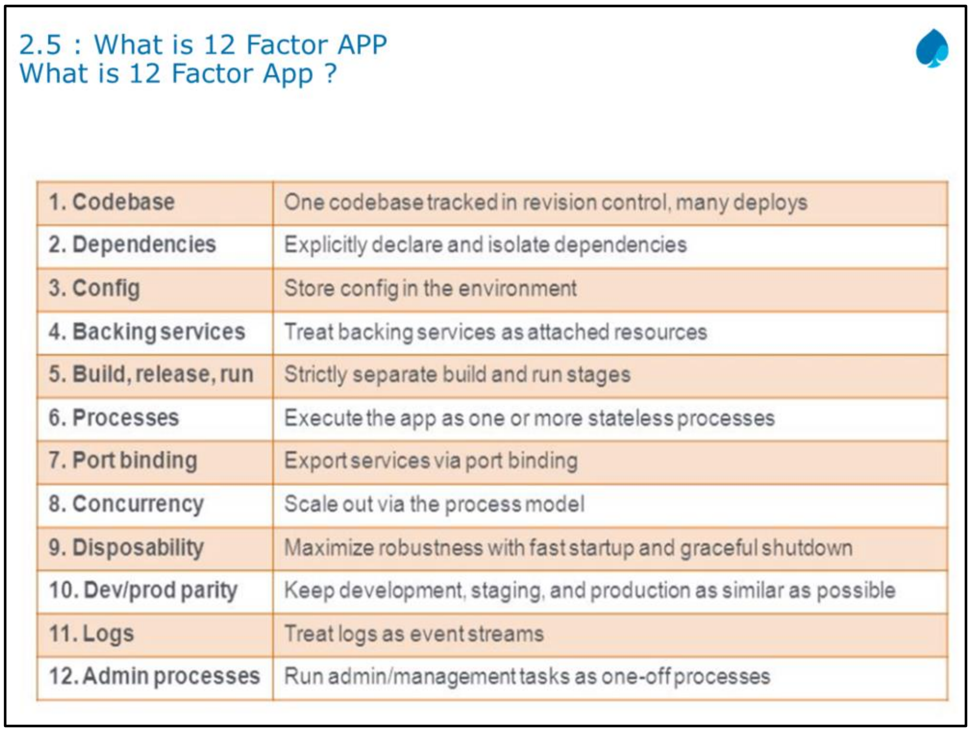
➢The 12-factor application methodology is to build Software-As-a-Service in Cloud.

Modern web applications run in heterogeneous environments, scale elastically, update frequently, and depend on independently deployed backing services. Modern application architectures and development practices must be designed accordingly. The PaaS-masters at Heroku summarized lessons learned from building hundreds of cloud-native applications into the twelve factors visualized below.

- The factors represent a set of guidelines or best practices for portable, resilient applications that will thrive in cloud environments (specifically software as a service applications .

This is For :--
• Any developer building applications which run as a service. •
- Ops engineers who deploy or manage such applications.

Instructor Notes:



2.5 : What is 12 Factor APP
What is 12 Factor App ?

1. Codebase	One codebase tracked in revision control, many deploys
2. Dependencies	Explicitly declare and isolate dependencies
3. Config	Store config in the environment
4. Backing services	Treat backing services as attached resources
5. Build, release, run	Strictly separate build and run stages
6. Processes	Execute the app as one or more stateless processes
7. Port binding	Export services via port binding
8. Concurrency	Scale out via the process model
9. Disposability	Maximize robustness with fast startup and graceful shutdown
10. Dev/prod parity	Keep development, staging, and production as similar as possible
11. Logs	Treat logs as event streams
12. Admin processes	Run admin/management tasks as one-off processes

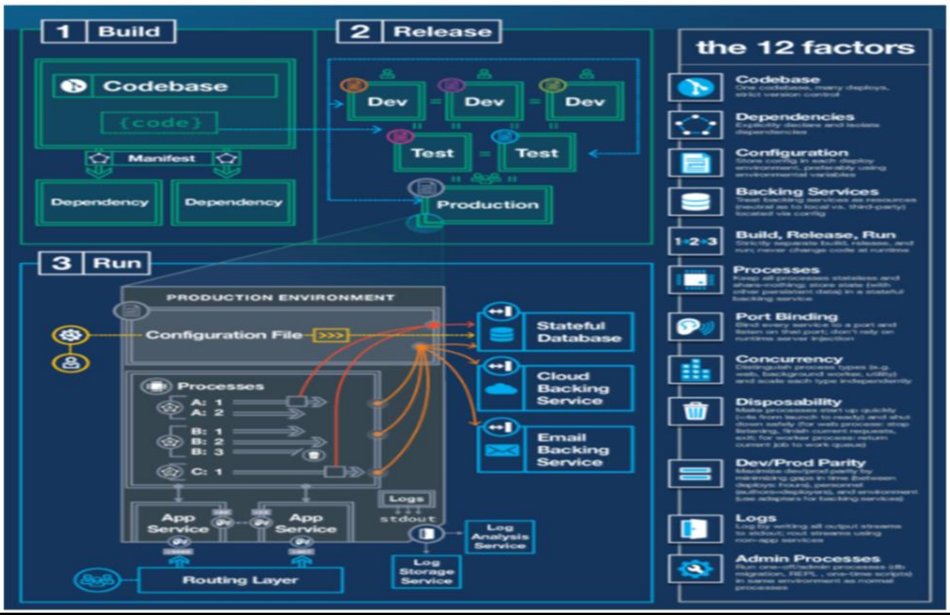
- The factors represent a set of guidelines or best practices for portable, resilient applications that will thrive in cloud environments (specifically software as a service applications .

This is For :--
• Any developer building applications which run as a service. •
- Ops engineers who deploy or manage such applications.

Instructor Notes:

2.5 : What is 12 Factor APP

What is 12 Factor App ?


1 Build
Codebase
{code}
Manifest
Dependency
Dependency
2 Release
Dev
Dev
Dev
Test
Test
Production
3 Run
PRODUCTION ENVIRONMENT
Configuration File
Processes
A: 1
A: 2
B: 1
B: 2
B: 3
C: 1
Logs
stdout
App Service
App Service
Routing Layer
Log Analysis Service
Log Storage Service
Stateful Database
Cloud Backing Service
Email Backing Service
the 12 factors
Codebase
One codebase, many deploys, strict version control
Dependencies
Explicitly declare and isolate dependencies
Configuration
Store config in each deploy environment, preferably using environmental variables
Backing Services
Treat backing services as resources (neutral as to local vs. third-party) located via config
Build, Release, Run
Strictly separate build, release, and run; never change code at runtime
Processes
Keep all processes stateless and share-nothing; store state (with other persistent data) in a stateful backing service
Port Binding
Bind every service to a port and listen on that port; don't rely on runtime server injection
Concurrency
Distinguish process types (e.g. web, background worker, utility) and scale each type independently
Disposability
Make processes start up quickly (<4s from launch to ready) and shut down safely (for web process: stop listening, finish current requests, exit; for worker process: return current job to work queue)
Dev/Prod Parity
Maximize dev/prod parity by minimizing gaps in time (between deploys: hours), personnel (authors=deployers), and environment (use adapters for backing services)
Logs
Log by writing all output streams to stdout; rout streams using non-app services
Admin Processes
Run one-off/admin processes (db migration, REPL, one-time scripts) in same environment as normal processes

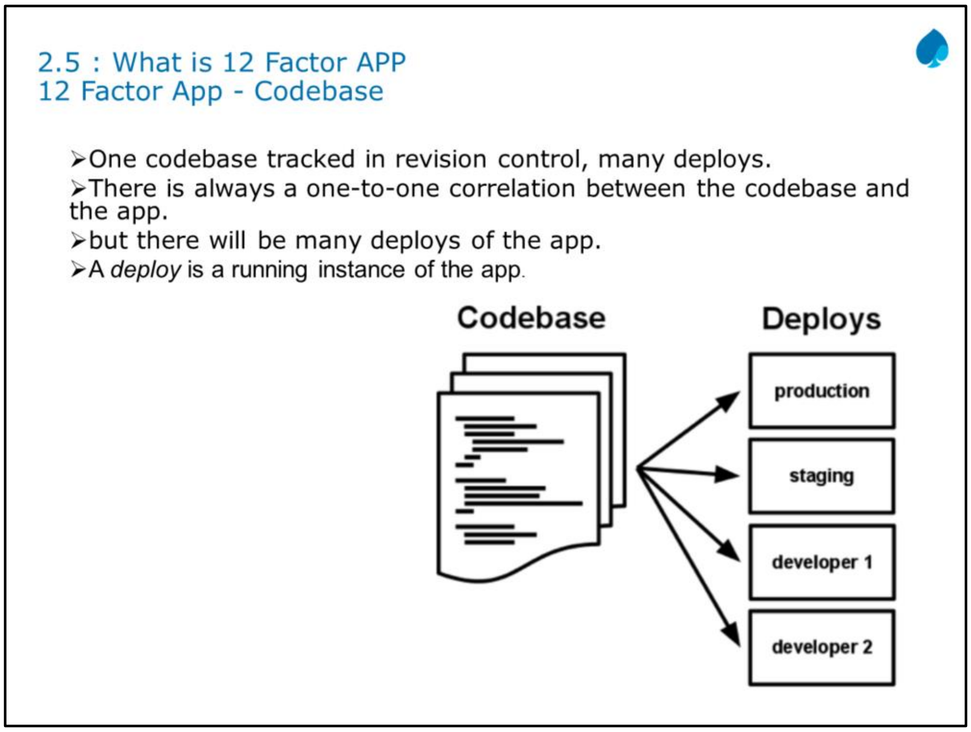
- The factors represent a set of guidelines or best practices for portable, resilient applications that will thrive in cloud environments (specifically software as a service applications .

This is For :--

• Any developer building applications which run as a service. •

- Ops engineers who deploy or manage such applications.

Instructor Notes:



2.5 : What is 12 Factor APP
12 Factor App - Codebase

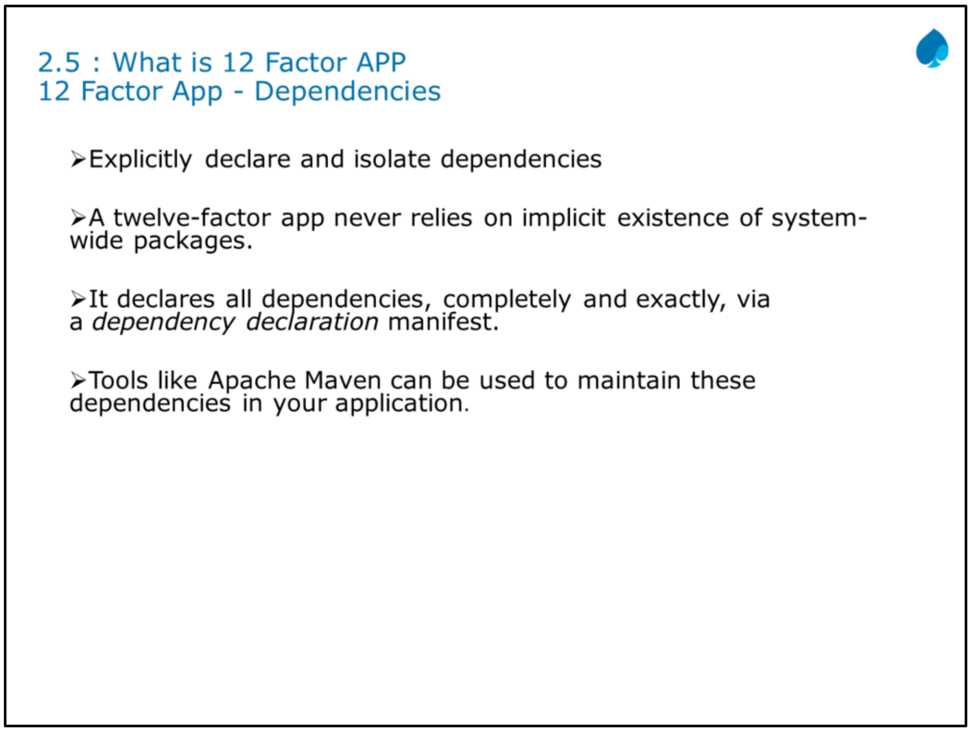
- ➢One codebase tracked in revision control, many deploys.
- ➢There is always a one-to-one correlation between the codebase and the app.
- ➢but there will be many deploys of the app.
- ➢A *deploy* is a running instance of the app.

Codebase
Deploys
production
staging
developer 1
developer 2

- A twelve-factor app is always tracked in a version control system, such as Git, Mercurial, or Subversion.
- A *codebase* is any single repo (in a centralized revision control system like Subversion), or any set of repos who share a root commit (in a decentralized revision control system like Git).
- A *deploy* is a running instance of the app. This is typically a production site, and one or more staging sites. Additionally, every developer has a copy of the app running in their local development environment, each of which also qualifies as a deploy.

- For example, a developer has some commits not yet deployed to staging; staging has some commits not yet deployed to production. But they all share the same codebase, thus making them identifiable as different deploys of the same app.

Instructor Notes:



2.5 : What is 12 Factor APP
12 Factor App - Dependencies

➢Explicitly declare and isolate dependencies

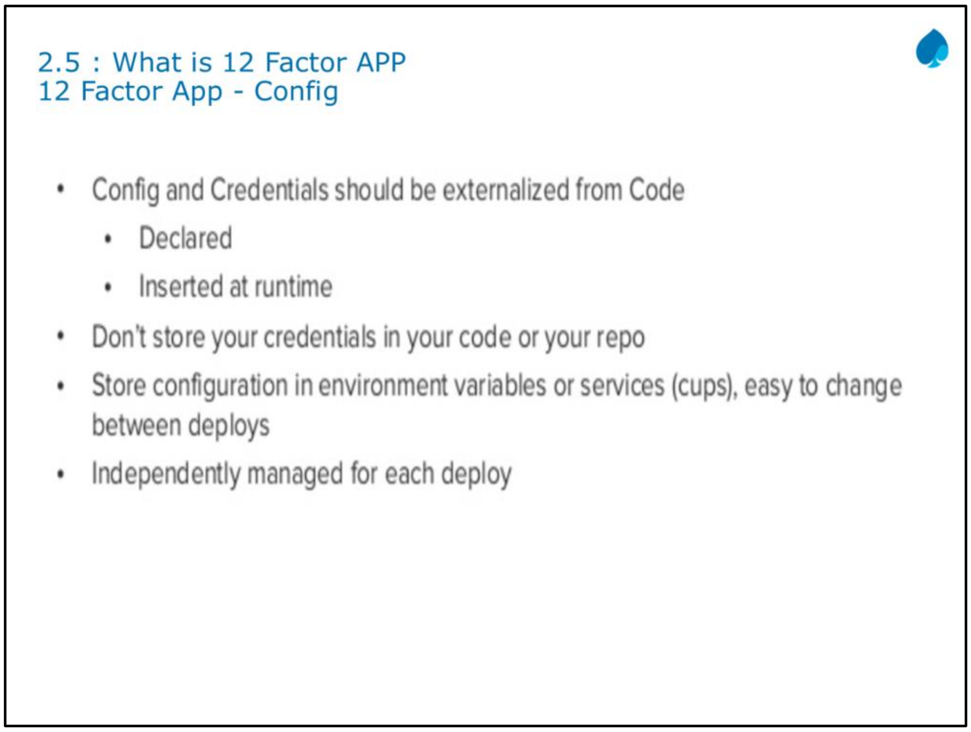
➢A twelve-factor app never relies on implicit existence of system-wide packages.

➢It declares all dependencies, completely and exactly, via a *dependency declaration* manifest.

➢Tools like Apache Maven can be used to maintain these dependencies in your application.

- Services should explicitly declare all **dependencies**, and should not rely on the presence of system-level tools or libraries: Main recommendation here is that avoid in pre-installed software in the system level which has a dependency with your application

Instructor Notes:

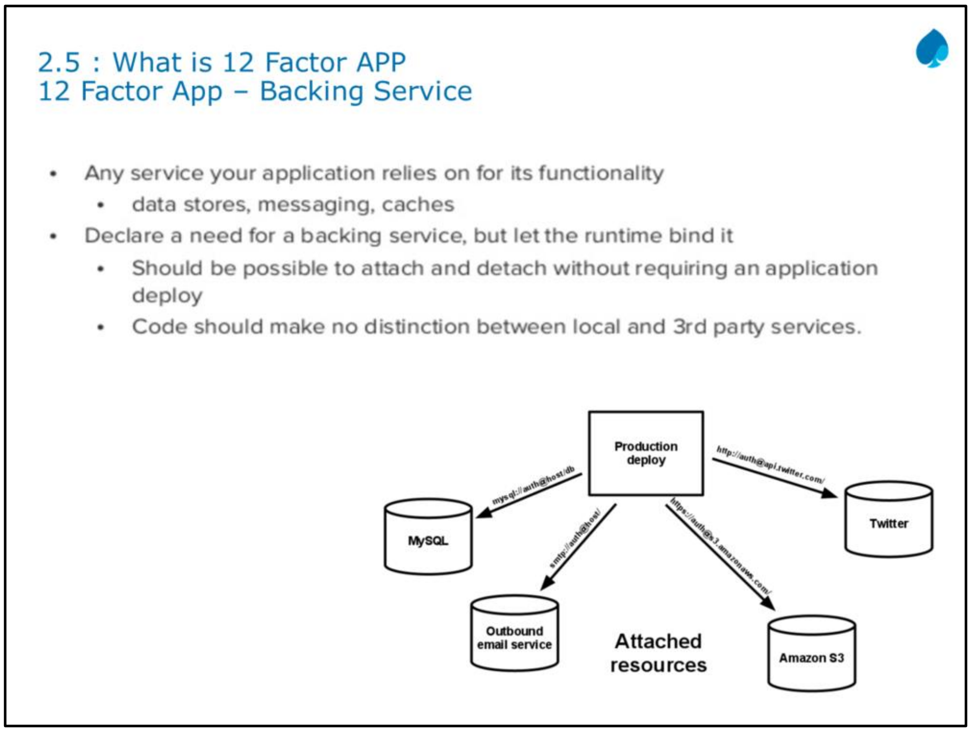


2.5 : What is 12 Factor APP
12 Factor App - Config

- Config and Credentials should be externalized from Code
  - Declared
  - Inserted at runtime
- Don't store your credentials in your code or your repo
- Store configuration in environment variables or services (cups), easy to change between deploys
- Independently managed for each deploy

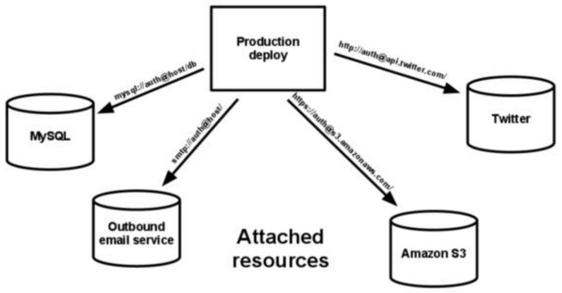
- **Configuration** that varies between deployment environments should be stored in the environment:
  - In this scenario the recommendation is to avoid having environment (Development, Staging and Production) specific configuration with in your application code.

Instructor Notes:



2.5 : What is 12 Factor APP
12 Factor App – Backing Service

- Any service your application relies on for its functionality
  - data stores, messaging, caches
- Declare a need for a backing service, but let the runtime bind it
  - Should be possible to attach and detach without requiring an application deploy
  - Code should make no distinction between local and 3rd party services.

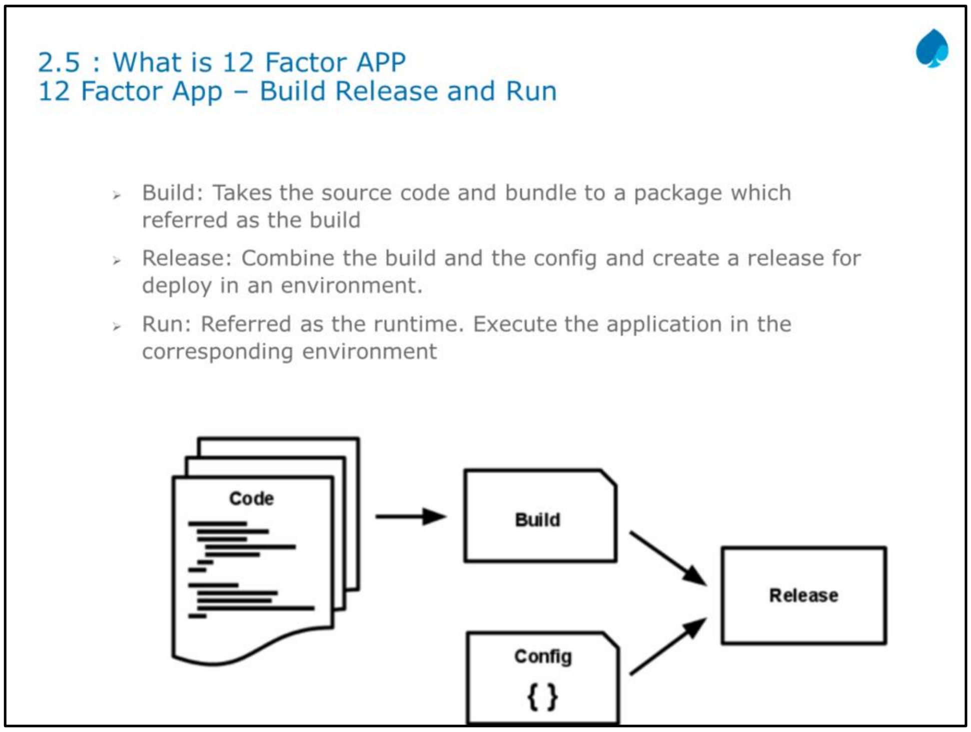


- A *backing service* is any service the app consumes over the network as part of its normal operation. Examples
  - include datastores (such as MySQL or CouchDB),
  - messaging/queueing systems (such as RabbitMQ or Beanstalk),
  - SMTP services for outbound email (such as Postfix),
  - and caching systems (such as Memcached).

The code for a twelve-factor app makes no distinction between local and third party services.
To the app, both are attached resources, accessed via a URL or other locator/credentials stored in the config.
A deploy of the twelve-factor app should be able to swap out a local MySQL database with one managed by a
third party (such as Amazon RDS) without any changes to the app's code.

For example, if the app's database is misbehaving due to a hardware issue, the app's administrator might spin up a new database server
restored from a recent backup. The current production database could be detached, and the new database attached – all without any code changes.

Instructor Notes:



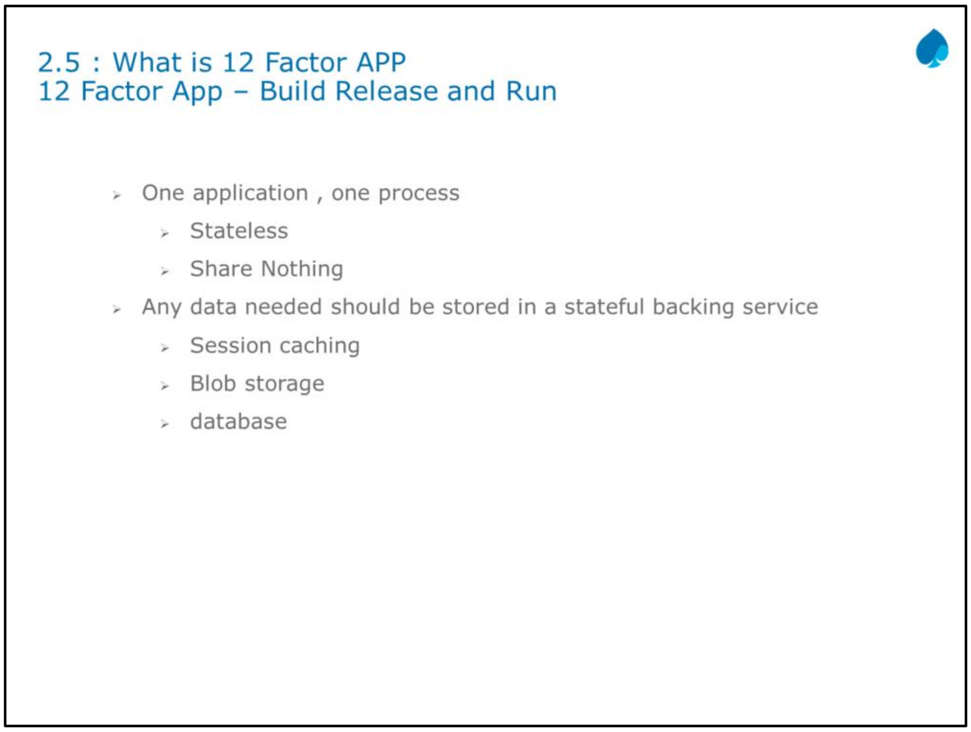
2.5 : What is 12 Factor APP
12 Factor App – Build Release and Run

- Build: Takes the source code and bundle to a package which referred as the build
- Release: Combine the build and the config and create a release for deploy in an environment.
- Run: Referred as the runtime. Execute the application in the corresponding environment

Code
Build
Config
{ }
Release

- Build: Takes the source code and bundle to a package which referred as the build
- Release: Combine the build and the config and create a release for deploy in an environment. Each release will have a unique identifier and related to a release management tool. It will ensure a quick rollback time.
- Run: Referred as the runtime. Execute the application in the corresponding environment

Instructor Notes:



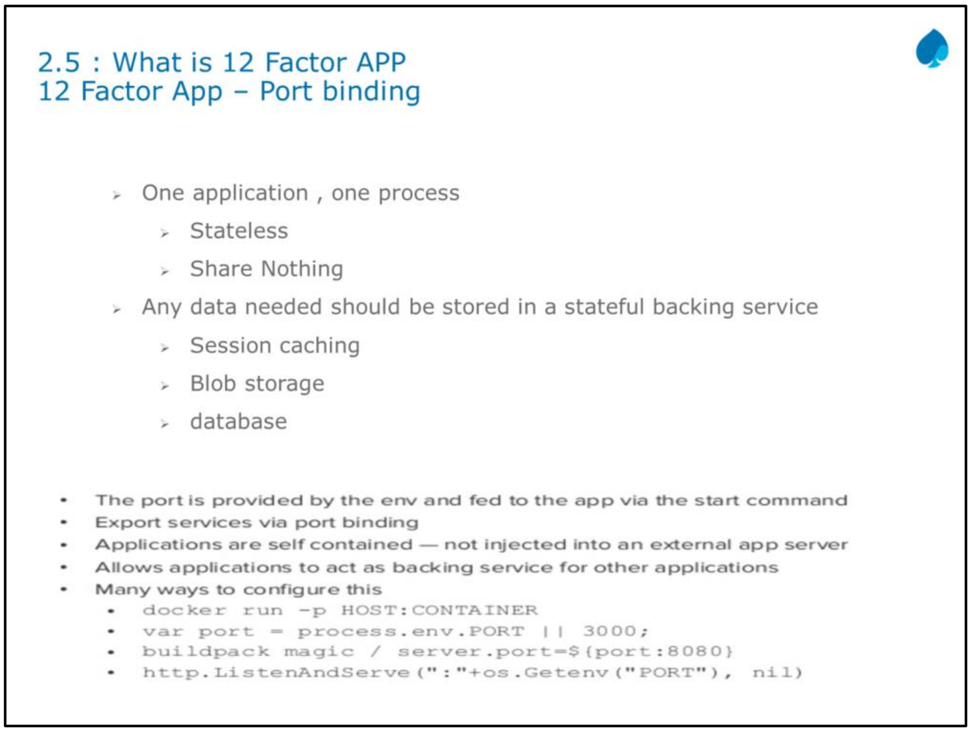
2.5 : What is 12 Factor APP
12 Factor App – Build Release and Run

- One application , one process
  - Stateless
  - Share Nothing
- Any data needed should be stored in a stateful backing service
  - Session caching
  - Blob storage
  - database

- **Twelve-factor processes are stateless and share-nothing.** Any data that needs to persist must be stored in a stateful backing service,

  typically a database.

Instructor Notes:



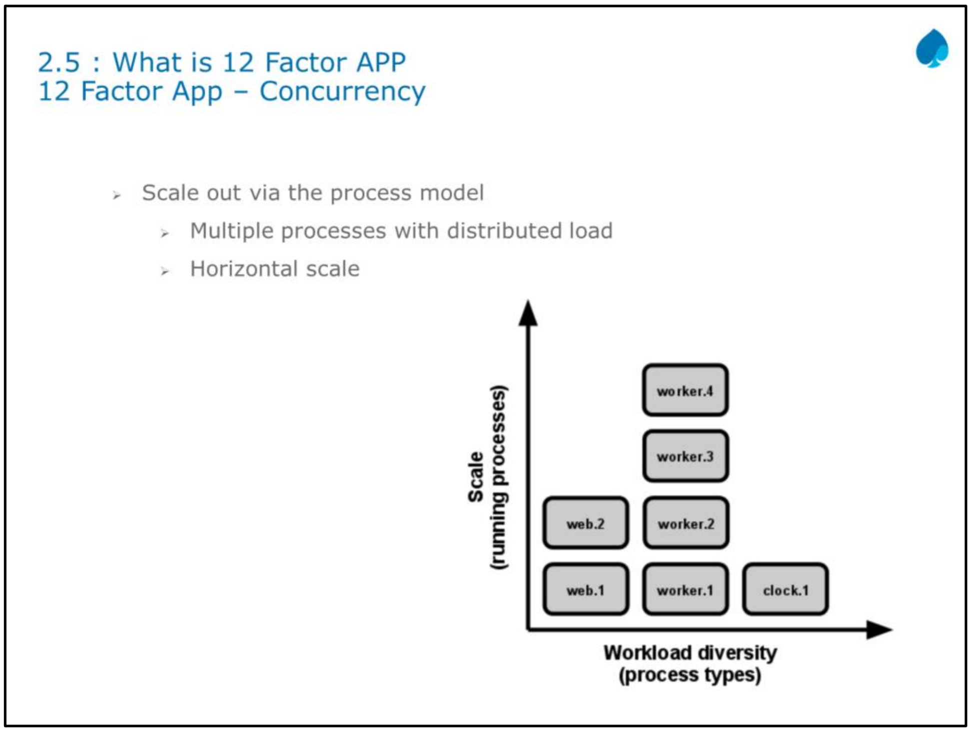
2.5 : What is 12 Factor APP
12 Factor App – Port binding

- One application , one process
  - Stateless
  - Share Nothing
- Any data needed should be stored in a stateful backing service
  - Session caching
  - Blob storage
  - database

- The port is provided by the env and fed to the app via the start command
- Export services via port binding
- Applications are self contained — not injected into an external app server
- Allows applications to act as backing service for other applications
- Many ways to configure this
  - `docker run -p HOST:CONTAINER`
  - `var port = process.env.PORT || 3000;`
  - `buildpack magic / server.port=${port:8080}`
  - `http.ListenAndServe(":"+os.Getenv("PORT"), nil)`

- Self-contained services should make themselves available to other services by **listening on a specified port**:
- This means each application are self-contained and expose access over a HTTP port that is bound to it.
- Spring boot framework is a good example for this, which is having a inbuilt server where you can configure the HTTP port easily

Instructor Notes:



2.5 : What is 12 Factor APP
12 Factor App – Concurrency

- Scale out via the process model
  - Multiple processes with distributed load
  - Horizontal scale

Scale
(running processes)
worker.4
worker.3
web.2
worker.2
web.1
worker.1
clock.1
Workload diversity
(process types)

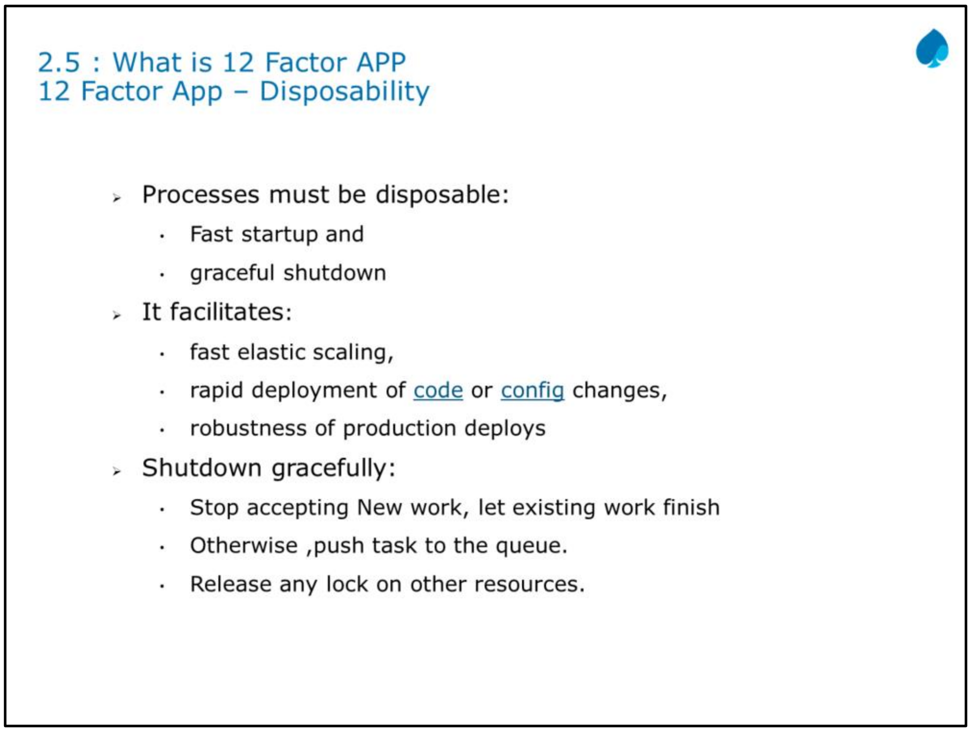
➢ Processes in the twelve-factor app take strong cues from the unix process model for running service daemons.

➢ Using this model, the developer can architect their app to handle diverse workloads by assigning each type of work to a *process type*.

➢ For example, HTTP requests may be handled by a web process, and long-running background tasks handled by a worker process.

➢ **Concurrency** is achieved by scaling individual processes (horizontal scaling): Idea behind this is having multiple processes with distributed load. The application should be able to scale horizontally and handle requests load-balanced to multiple identical running nodes of the application. In addition application should be able to scale out processes or threads for parallel execution of work in an on-demand basis. This feature comes automatically with the JVM with multi-threading.

Instructor Notes:

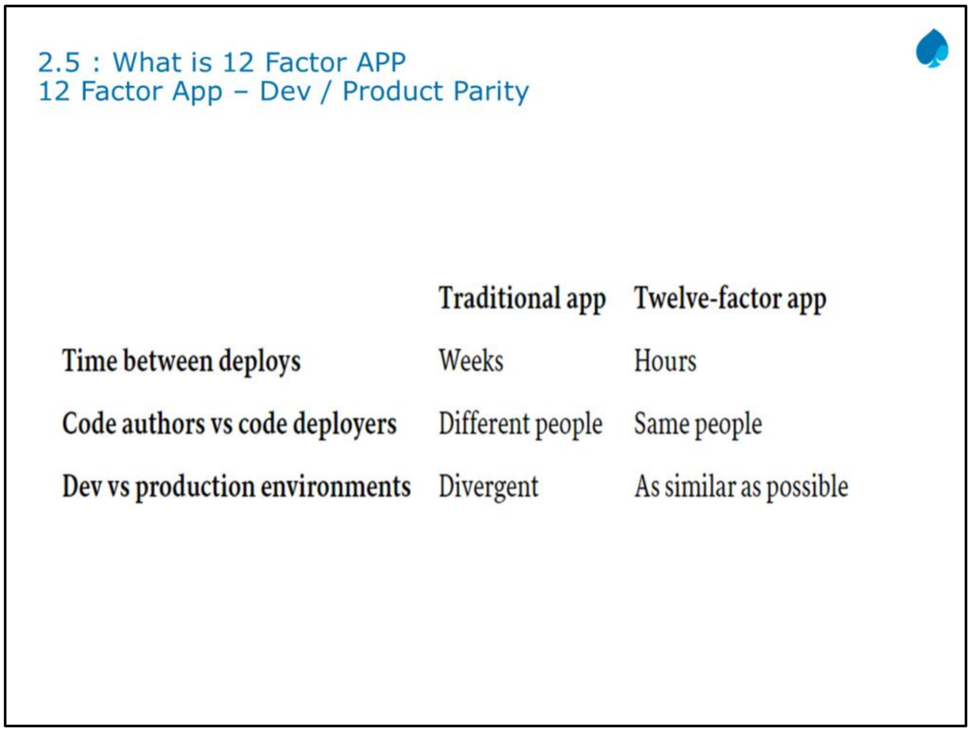


2.5 : What is 12 Factor APP
12 Factor App – Disposability

- Processes must be disposable:
  - Fast startup and
  - graceful shutdown
- It facilitates:
  - fast elastic scaling,
  - rapid deployment of code or config changes,
  - robustness of production deploys
- Shutdown gracefully:
  - Stop accepting New work, let existing work finish
  - Otherwise ,push task to the queue.
  - Release any lock on other resources.

- Our application should minimize the startup time like using backing services rather than using in-memory caching.
- Also in shutdown process, when we stop the application it should not accept new work and

Instructor Notes:



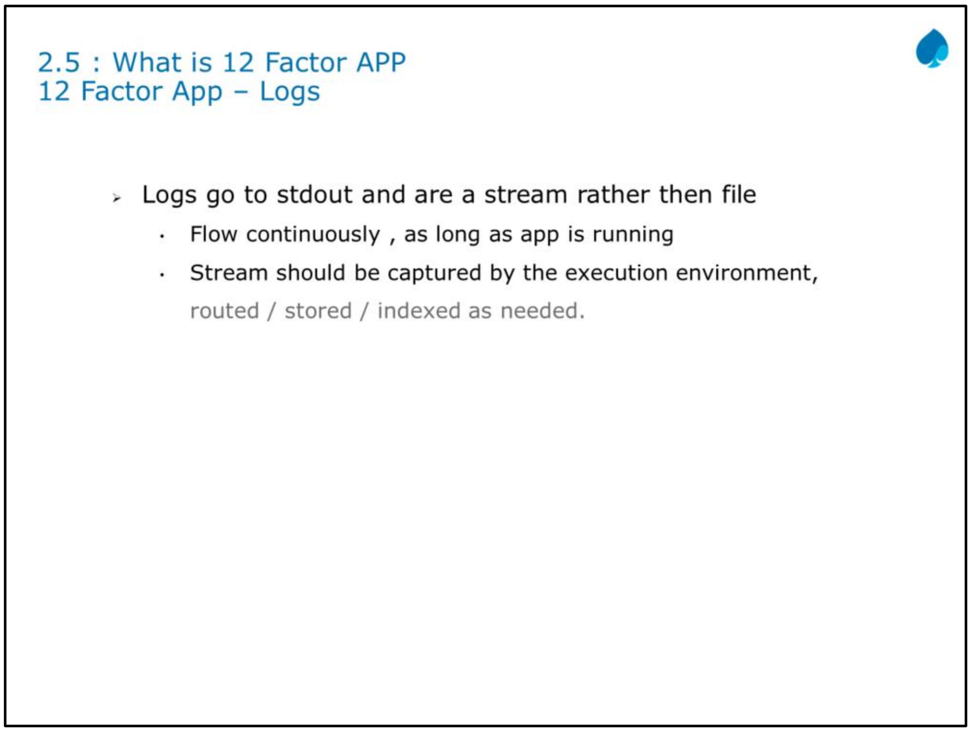
2.5 : What is 12 Factor APP
12 Factor App – Dev / Product Parity

	Traditional app	Twelve-factor app
Time between deploys	Weeks	Hours
Code authors vs code deployers	Different people	Same people
Dev vs production environments	Divergent	As similar as possible

- All **environments**, from local development to production, should be as similar as possible:

- **The twelve-factor app is designed for continuous deployment by keeping the gap between development and production small.**
- Looking at the three gaps described above:
  - Make the time gap small: a developer may write code and have it deployed hours or even just minutes later.
  - Make the personnel gap small: developers who wrote code are closely involved in deploying it and watching its behavior in production.
  - Make the tools gap small: keep development and production as similar as possible.

Instructor Notes:

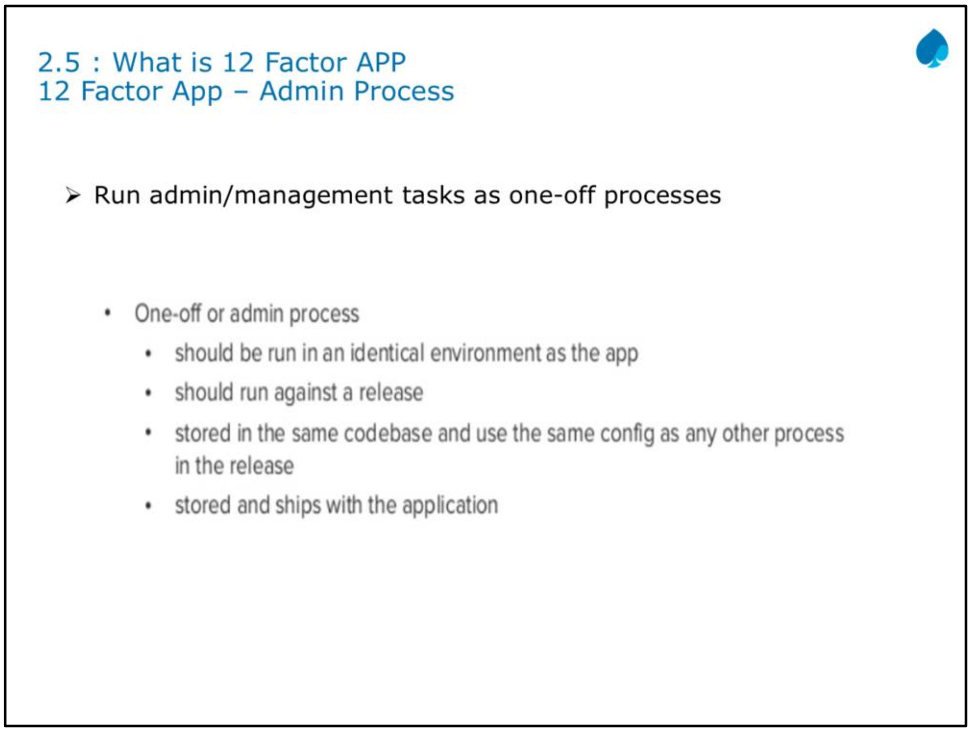


2.5 : What is 12 Factor APP
12 Factor App – Logs

- Logs go to stdout and are a stream rather then file
  - Flow continuously , as long as app is running
  - Stream should be captured by the execution environment, routed / stored / indexed as needed.

- **A twelve-factor app never concerns itself with routing or storage of its output stream.**
- It should not attempt to write to or manage logfiles. Instead, each running process writes its event stream, unbuffered, to stdout.
- During local development, the developer will view this stream in the foreground of their terminal to observe the app's behavior.

Instructor Notes:

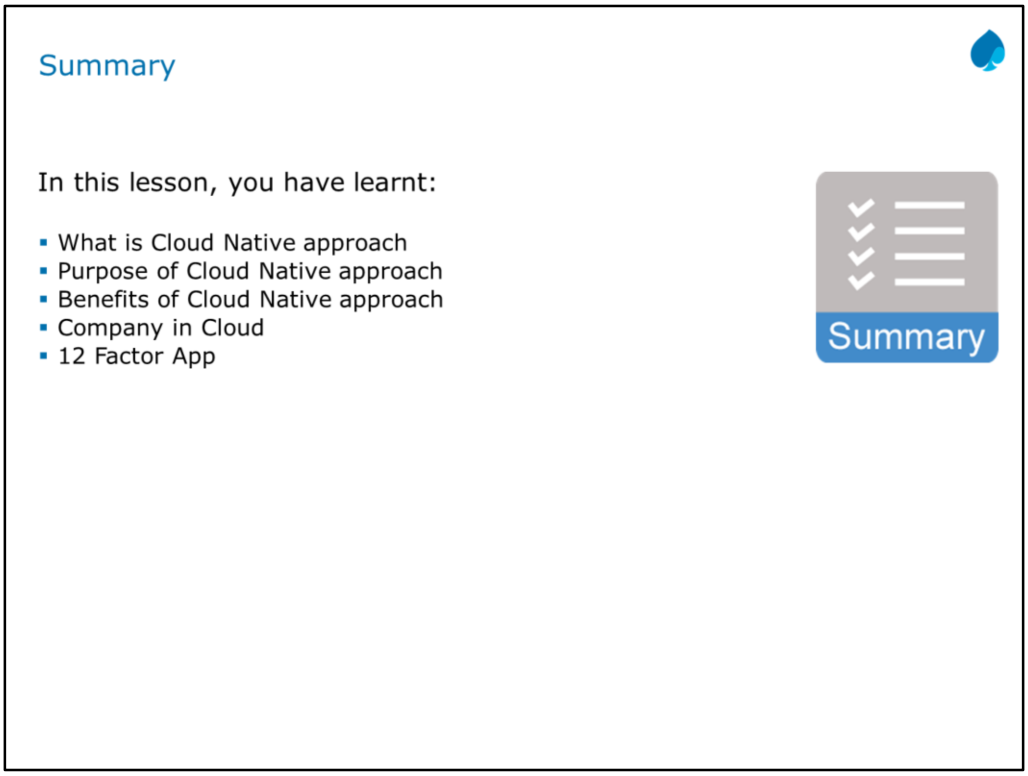


2.5 : What is 12 Factor APP
12 Factor App – Admin Process

- Run admin/management tasks as one-off processes
  - One-off or admin process
    - should be run in an identical environment as the app
    - should run against a release
    - stored in the same codebase and use the same config as any other process in the release
    - stored and ships with the application

- **A twelve-factor app never concerns itself with routing or storage of its output stream.**
- It should not attempt to write to or manage logfiles. Instead, each running process writes its event stream, unbuffered, to stdout.
- During local development, the developer will view this stream in the foreground of their terminal to observe the app’s behavior.

Instructor Notes:



Summary

In this lesson, you have learnt:

- What is Cloud Native approach
- Purpose of Cloud Native approach
- Benefits of Cloud Native approach
- Company in Cloud
- 12 Factor App

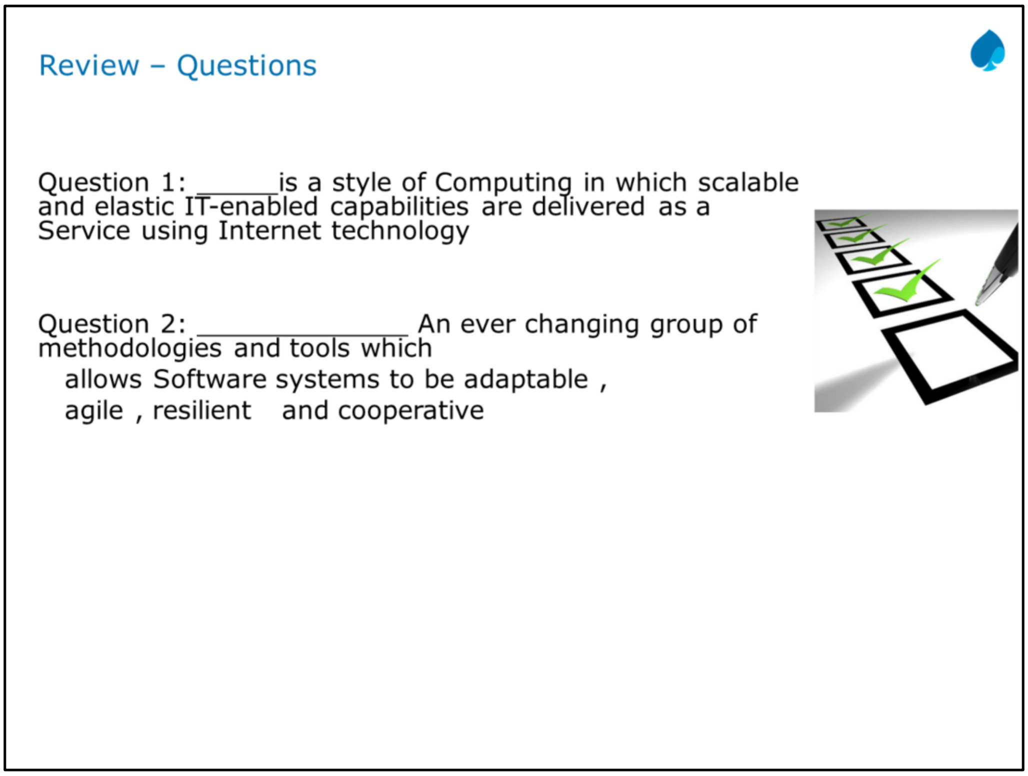
Summary

Instructor Notes:

Answers for the Review Questions:

Answer 1: Cloud

Answer 2: Cloud Native



Review – Questions

Question 1: ______is a style of Computing in which scalable and elastic IT-enabled capabilities are delivered as a Service using Internet technology

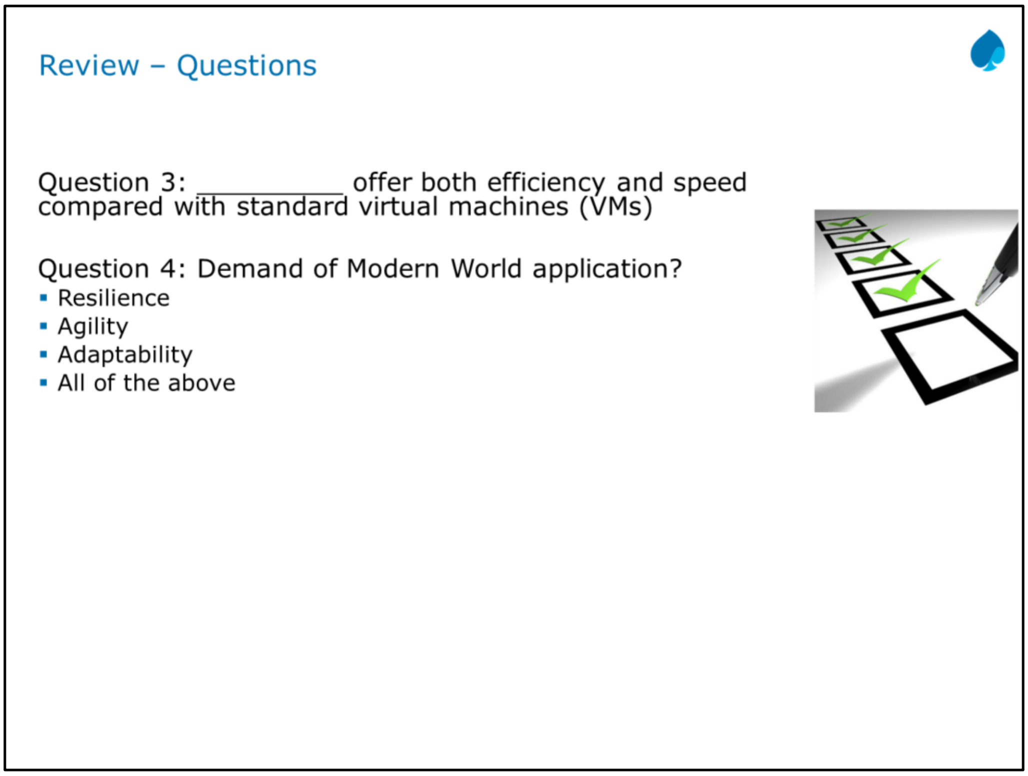
Question 2: _______________ An ever changing group of methodologies and tools which allows Software systems to be adaptable , agile , resilient and cooperative

Instructor Notes:

Answers for the Review Questions:

Answer 3: Container

Answer 4: All of the above



Review – Questions

Question 3: __________ offer both efficiency and speed compared with standard virtual machines (VMs)

Question 4: Demand of Modern World application?

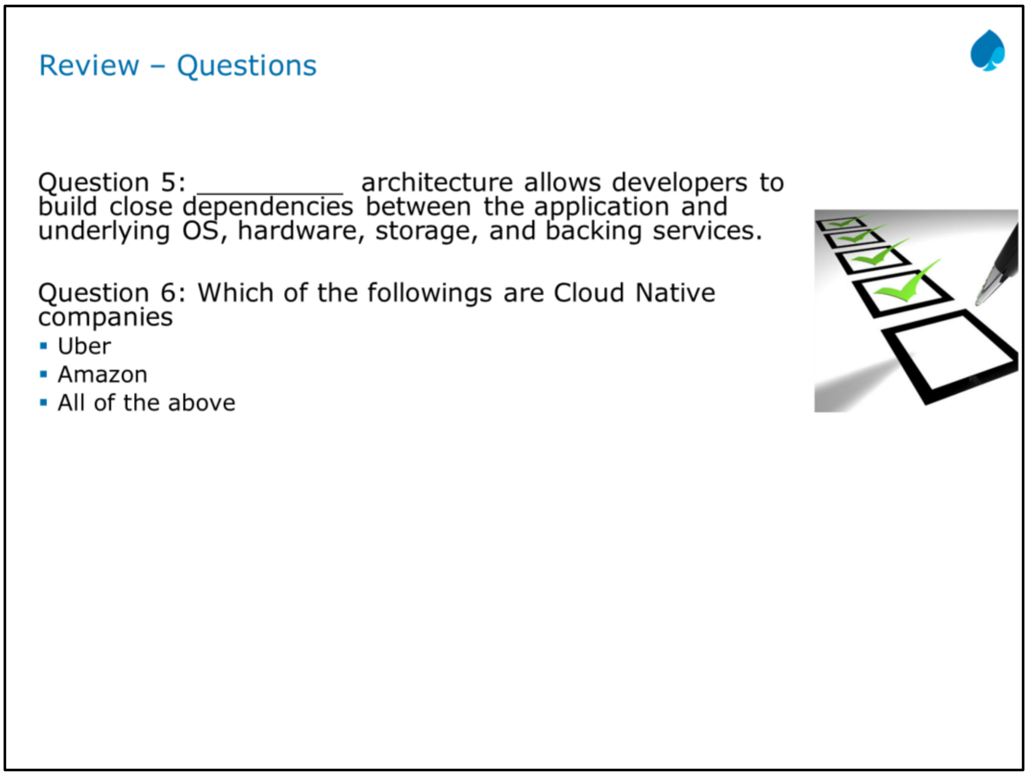
- Resilience
- Agility
- Adaptability
- All of the above

Instructor Notes:

Answers for the Review Questions:

Answer 5: Traditional

Answer 6: All of the above



Review – Questions

Question 5: ___________ architecture allows developers to build close dependencies between the application and underlying OS, hardware, storage, and backing services.

Question 6: Which of the followings are Cloud Native companies

- Uber
- Amazon
- All of the above