Cloud Computing V2

Cloud-native versus cloud-enabled

A cloud-enabled application is an application that was developed for deployment in a traditional data center but was later changed so that it also could run in a cloud environment. However, cloud-native applications are built to operate only in the cloud. Developers design cloud-native applications to be scalable, platform-neutral, and composed of microservices.

Cloud-native development methods

When developing cloud-native applications, developers must understand and adopt new methods and patterns to maximize the capability that is provided by the cloud provider:

- Provides readily available sandbox and production environments. These environments offer the following capabilities that are attractive for developers:
- Free trials that are offered with most products.
- Pre-built templates and examples that help developers to get started fast

- Easier to understand the application lifecycle.

- Brings a wide range of choices to developers in the following areas:
- Programming languages and frameworks.

- Services.
- APIs.

- A developer toolchain facilitates integrated development, test, and debugging:
- The new model is to integrate development and operations teams into DevOps.
- Build engine for compilation and testing.

Cloud-native development introduces the 12-factor app methods and patterns to development:

- Factor 1. One codebase that is tracked in revision control with multiple deployments: Use one codebase for tracking and revision control, and from that codebase, you can deploy many times.
- Factor 2. Explicitly declare and isolate dependencies. For example, using a package.json file for a Node.js application lists all the external dependencies.
- Factor 3. Store configuration in the environment: Provide any extra configuration information as environment variable that can be bootstrapped when the application starts.
- Factor 4. Treat backing services as attached resources: Bind services to applications by using attach services (such as DBaaS or load balancers) to an application.
- Factor 5. Strictly separate build and run stages: For example, it is impossible to make changes to the code at run time, since there is no way to propagate those changes back to the build stage.
- Factor 6. Run the app as one or more stateless processes: When you design applications, use multiple processes or services as needed. Avoid dependencies on sticky sessions and keep session data in a persistent store to ensure

traffic can be routed to other processes without service disruption.

environment and binds to this port.

- Factor 8. Scale out by using the process model: Horizontal scaling of application instances in cloud providers.
- Factor 9. Maximize robustness with fast startup and graceful shutdown: Use a disposable approach to the design of a process in the application. There should be minimal startup actions required. When a process is terminated, it should exit with minimal housekeeping, which improves robustness and responsiveness to horizontal scaling events.
- Factor 10. Keep development, staging, and production similar: This approach enables agile software delivery and continuous integration.
- Factor 11. Treat logs as event streams: Cloud providers have ways to accumulate log data across various components of the application to enable analyzing or exporting to a third-party logging service.
- Factor 12. Run administrator and management tasks as one-off processes: Design tasks that must run once or occasionally as separate components that can be run when needed instead of adding the code directly into another component. For example, if an application must migrate data into a database, place this task into a separate component instead of adding it to the main application code at startup.

← Previous *Completed* 17 of 21 *Modules* Next →