100 XP

Explore Serverless computing in Azure

7 minutes

Serverless computing is the abstraction of servers, infrastructure, and OSs. With *serverless* computing, Azure takes care of managing the server infrastructure and allocation/deallocation of resources based on demand. Infrastructure isn't your responsibility. Scaling and performance are handled automatically, and you are billed only for the exact resources you use. There's no need to even reserve capacity.

Serverless computing encompasses three ideas: the abstraction of servers, an event-driven scale, and micro-billing:

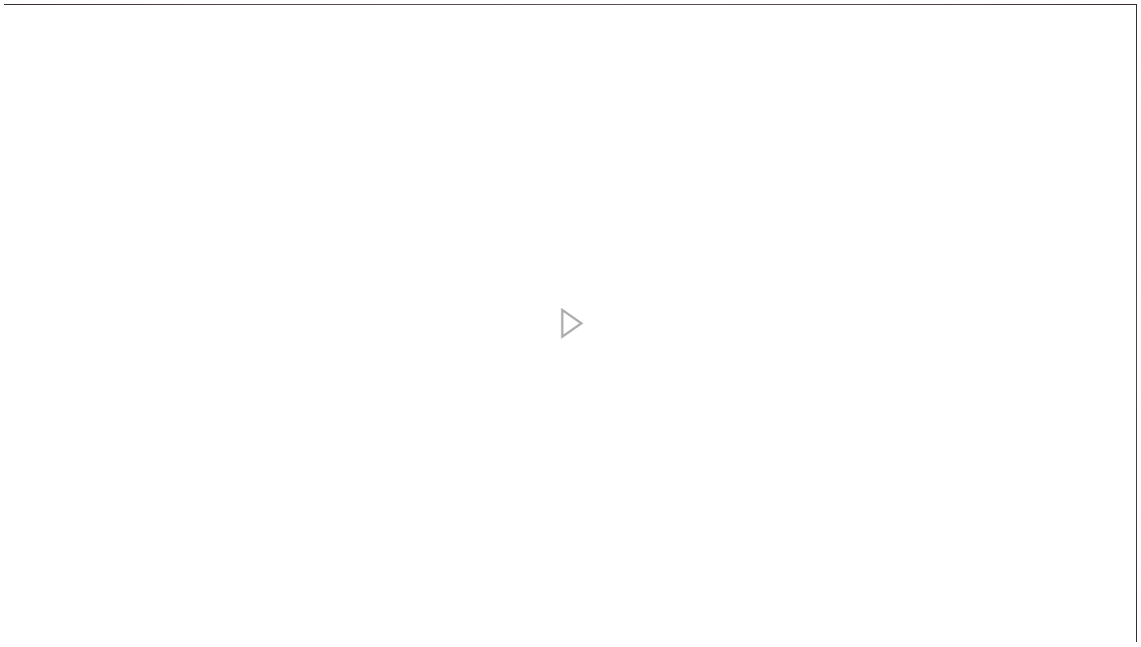
1. **Abstraction of servers**: Serverless computing abstracts the servers you run on. You never explicitly reserve server instances; the platform manages that for you. Each function execution can run on a different compute instance, and this execution context is transparent to the code. With serverless architecture, you simply deploy your code, which then runs with high availability.

2. **Event-driven scale**: Serverless computing is an excellent fit for workloads that respond to incoming events. Events include triggers by timers (for example, if a function needs to run every day at 10:00 AM UTC), HTTP (API and webhook scenarios), queues (for example, with order processing), and much more. Instead of writing an entire application, the developer authors a function, which contains both code and metadata about its triggers and bindings. The platform automatically schedules the function to run and scales the number of compute instances based on the rate of incoming events. Triggers define how a function is invoked and bindings provide a declarative way to connect to services from within the code.

3. **Micro-billing**: Traditional computing has the notion of per-second billing, but often, that's not as useful as it seems. Even if a customer's website gets only one hit a day, they still pay for a full day's worth of availability. With serverless computing, they pay only for the time their code runs. If no active function executions occur,

they're not charged. For example, if the code runs once a day for two minutes, they're charged for one execution and two minutes of computing time.

Serverless computing in Azure



Azure has two implementations of serverless compute:

- **Azure Functions**, which can execute code in almost any modern language.
- **Azure Logic Apps**, which are designed in a web-based designer and can execute logic triggered by Azure services without writing any code.

Azure Functions

When you're concerned only about the code running your service, and not the underlying platform or infrastructure, Azure Functions are ideal. They're commonly used when you need to perform work in response to an event, often via a REST request, timer, or message from another Azure service and when that work can be completed quickly, within seconds or less.

Azure Functions scale automatically based on demand, so they're a solid choice when demand is variable. For example, you may be receiving messages from an IoT solution used to monitor a fleet of delivery vehicles. You'll likely have more data arriving during business hours.

Using a VM-based approach, you'd incur costs even when the VM is idle. With functions, Azure runs your code when it's triggered and automatically deallocates resources when the function is finished. In this model, you're only charged for the CPU time used while your function runs.

Furthermore, Azure Functions can be either stateless (the default), where they behave as if they're restarted every

time they respond to an event, or stateful (called "Durable Functions"), where a context is passed through the function to track prior activity.

Functions are a key component of serverless computing, but they're also a general compute platform for running any type of code. If the needs of the developer's app change, you can deploy the project in an environment that isn't serverless, which provides the flexibility to manage scaling, run on virtual networks, and even completely isolate the functions.

Azure Logic Apps

Azure Logic Apps are similar to Functions - both enable you to trigger logic based on an event. Where Functions execute code, Logic Apps execute *workflows* designed to automate business scenarios and built from predefined logic blocks. Every logic app workflow starts with a trigger, which fires when a specific event happens or when newly available data meets specific criteria. Many triggers include basic scheduling capabilities, so developers can specify how regularly their workloads will run. Each time the trigger fires, the Logic Apps engine creates a logic app instance that runs the actions in the workflow. These actions can also include data conversions and flow controls, such as conditional statements, switch statements, loops, and branching.

You create Logic App workflows using a visual designer on the Azure portal or in Visual Studio. The workflows are persisted as a JSON file with a known workflow schema.

Azure provides over 200 different connectors and processing blocks to interact with different services - including most popular enterprise apps. You can also build custom connectors and workflow steps if the service you need to interact with isn't covered. You then use the visual designer to link connectors and blocks together, passing data through the workflow to do custom processing - often all without writing any code.

As an example, let's say a ticket arrives in ZenDesk. You could:

1. Detect the intent of the message with cognitive services
2. Create an item in SharePoint to track the issue
3. If the customer isn't in your database, add them to your Dynamics 365 CRM system
4. Send a follow-up email to acknowledge their request

All of that could be designed in a visual designer making it easy to see the logic flow, which is ideal for a business analyst role.

Functions vs. Logic Apps

Functions and Logic Apps can both create complex orchestrations. An orchestration is a collection of functions or steps, that are executed to accomplish a complex task. With Azure Functions, you write code to complete each step, with Logic Apps, you use a GUI to define the actions and how they relate to one another.

You can mix and match services when you build an orchestration, calling functions from logic apps and calling logic

apps from functions. Here are some common differences between the two.

-	**Functions**	**Logic Apps**
State	Normally stateless, but Durable Functions provide state	Stateful
Development	Code-first (imperative)	Designer-first (declarative)
Connectivity	About a dozen built-in binding types, write code for custom bindings	Large collection of connectors, Enterprise Integration Pack for B2B scenarios, build custom connectors
Actions	Each activity is an Azure function; write code for activity functions	Large collection of ready-made actions
Monitoring	Azure Application Insights	Azure portal, Log Analytics
Management	REST API, Visual Studio	Azure portal, REST API, PowerShell, Visual Studio
Execution context	Can run locally or in the cloud	Runs only in the cloud.

Next unit: Summary

Continue >