# Module 5: Authoring Serverless Applications in Azure

## Contents:



## Module overview

This module describes how solutions can leverage serverless application hosting services in Azure to host web applications, REST APIs, integration workflows and HPC workloads without the requirement to manage specific server resources. The module focuses on App Services-related components such as Web Apps, API Apps, Mobile Apps, Logic Apps, and Functions.

### Objectives

After completing this module, students will be able to:

- Select between hosting application code or containers in an App Service instance.
- Describe the differences between API, Mobile, and Web Apps.
- Integrate an API or Logic App with the API Management service.
- Design an App Service Plan or multi-region deployment for high performance and scale.

## Lesson 1: Azure Web Apps

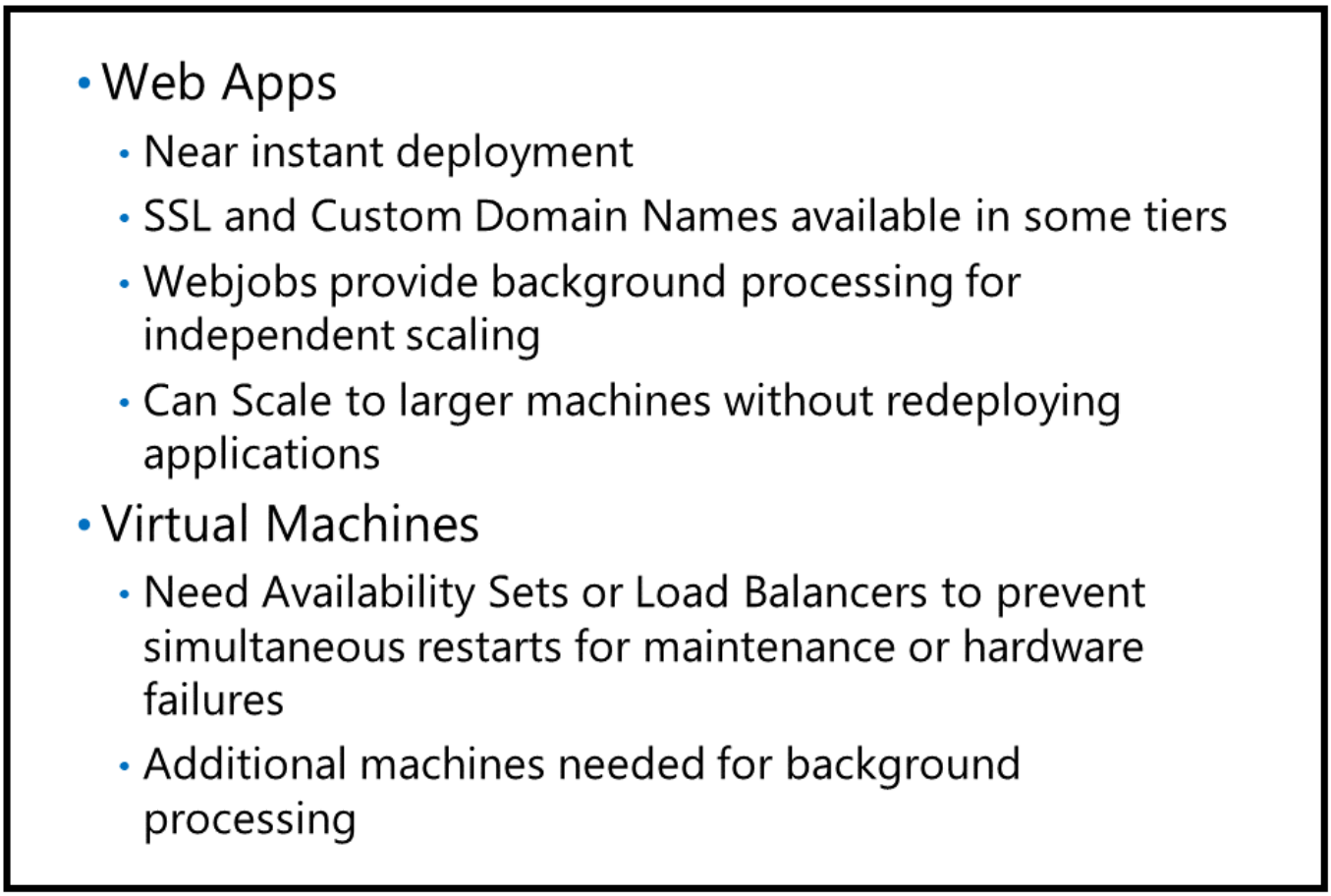
This lesson describes the Web Apps service. In many scenarios, it is preferable to use a quick and easy way to deploy web applications to the cloud rather than to reengineer the web applications as cloud projects. Web Apps allow you to quickly create a new Web App and iterate changes to the Web App in an agile manner.

## Lesson objectives

After completing this lesson, you will be able to:

- Describe the Web Apps service.
- List the different tiers for a Web App.

## Web Apps




Web Apps is a low friction Platform-as-a-Service (PaaS) offering to host your web applications in the Azure platform. The service is fully managed and you can easily configure advanced features such as AlwaysOn, custom domains, and autoscale by using either portal.

**Flexibility**

You can use a variety of integrated development environments (IDEs) and frameworks, such as .NET, Java, PHP, Node.js, or Python, to develop your web applications that are eventually deployed to Azure Web Apps. You can use Git and Kudu to deploy Node.js or PHP web applications. You also can deploy web applications that are developed in Microsoft Visual Studio to Web Apps by using the File Transfer Protocol (FTP) or the Web Deploy protocol.

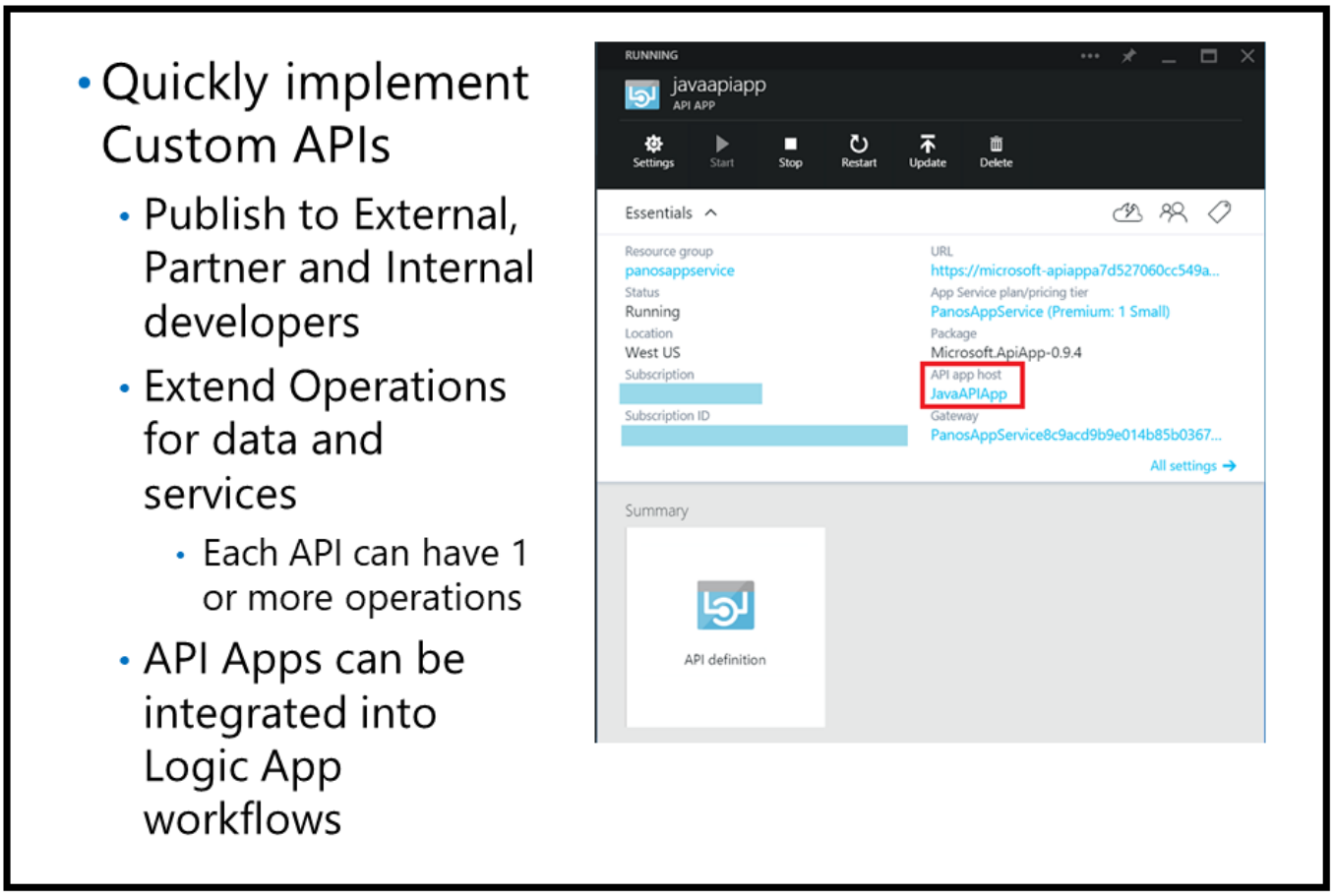
**Scalability**

Because Web Apps is a fully managed service implementation, you can focus on developing your application and solving business problems instead of the hosting implementation and hardware scaling or specifics. You can easily scale up a stateless web application by configuring autoscale in the portal. Autoscale creates multiple instances of your Web App that are automatically load balanced so that your application can meet potential spikes in demand.

### Web App Containers

The Web Apps service is offered in both a Windows and Linux variant. The Linux variant specifically offers the ability to host Docker containers directly using a Web App. The docker containers can be sourced from Docker Hub, Azure Container Registry or GitHub. Containers can be deployed manually, as part of the Web App deployment or deployed in a streamlined continuous integration process using Docker Hub or GitHub.

## API Apps




API and Mobile Apps are specialized version of Web Apps designed to server different purposes in an all-up solution on Azure.

### API Apps

API Apps provide specific support for developing, hosting and securing your custom API's in the context of App. API Apps can run either custom code or can run dozens of pre-built software to connect to existing popular Software-as-a-Service solutions. API Apps share a lot of the same features and support as Web Apps.

### Mobile Apps

Azure Mobile Apps is a component of Azure App Services offering designed to make it easy to create highly-functional mobile apps using Azure. Mobile Apps brings together a set of Azure services that enable backend capabilities for your apps. Mobile Apps provides the following backend capabilities in Azure to support your apps:

- **Single Sign On** - Select from an ever-growing list of identity providers including Azure Active Directory, Facebook, Google, Twitter, and Microsoft Account, and leverage Mobile Apps to add authentication to your app in minutes.

- **Offline Sync** - Mobile Apps makes it easy for you to build robust and responsive apps that allow employees to work offline when connectivity is not available, and synchronize with your enterprise backend systems when devices comes back online. Offline sync capability is supported on all client platforms and works with any data source including SQL, Table Storage, Mongo, or Document DB, and any SaaS API including Office 365, Salesforce, Dynamics, or on-premises databases.
- **Push Notifications** - Mobile Apps offers a massively scalable mobile push notification engine, Notification Hubs, capable of sending millions of personalized push notifications to dynamic segments of audience using iOS, Android, Windows, or Kindle devices within seconds. You can easily hook Notification Hubs to any existing app backend, whether that backend is hosted on-premises or in the cloud.
- **Auto Scaling** - App Service enables you to quickly scale-up or out to handle any incoming customer load. Manually select the number and size of VMs or set up auto-scaling to scale your mobile app backend based on load or schedule.

Mobile App endpoints are, at their simplest, REST APIs and can be used on a wide variety of platforms and with a wide variety of devices. Client SDKs are available, however, if you like to connect your mobile application to a Mobile App instance for its backend data.

Mobile client SDKs are available for the following platforms:

- Xamarin Android/IOS
- Android Native
- IOS Native
- Windows Store
- Windows Phone
- .NET
- HTML

## Lesson 2: Azure Functions

This lesson briefly introduces Azure Functions and the concept of Serverless processing.

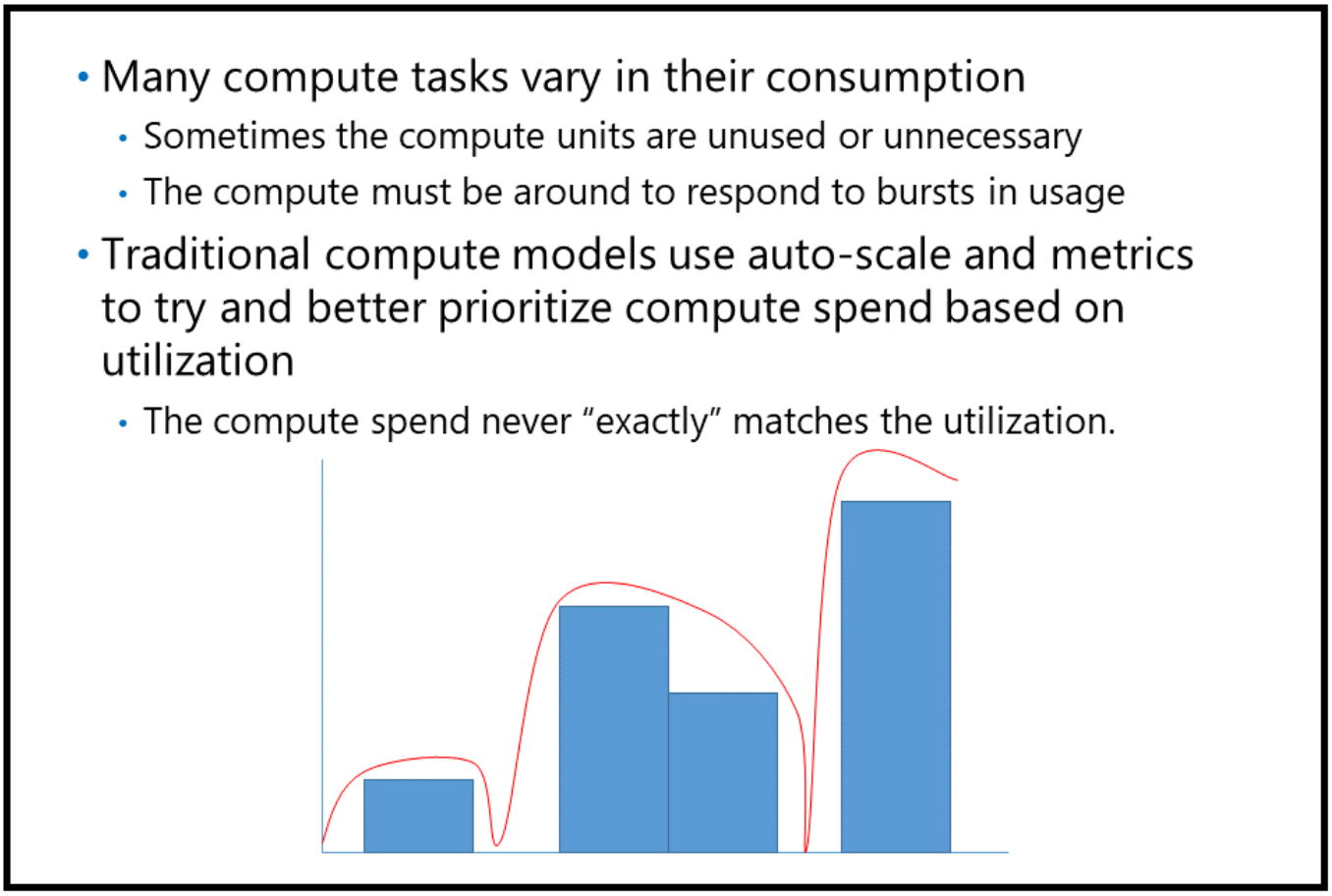
### Lesson objectives

After completing this lesson, you will be able to:

- Explain how serverless compute can potentially save costs

- Describe the Azure Functions service

## Serverless Processing




Azure Functions are the newest kind of App available in App Services. Function Apps are designed to make it faster to process events and distribute output to bound services.

> **Note:** Function Apps were implemented using the existing code and base functionality for Azure WebJobs. To this end, you can go to GitHub and view the Azure WebJobs repositories to see the open-source implementation of Function Apps.

Function Apps can be created using one of two models:

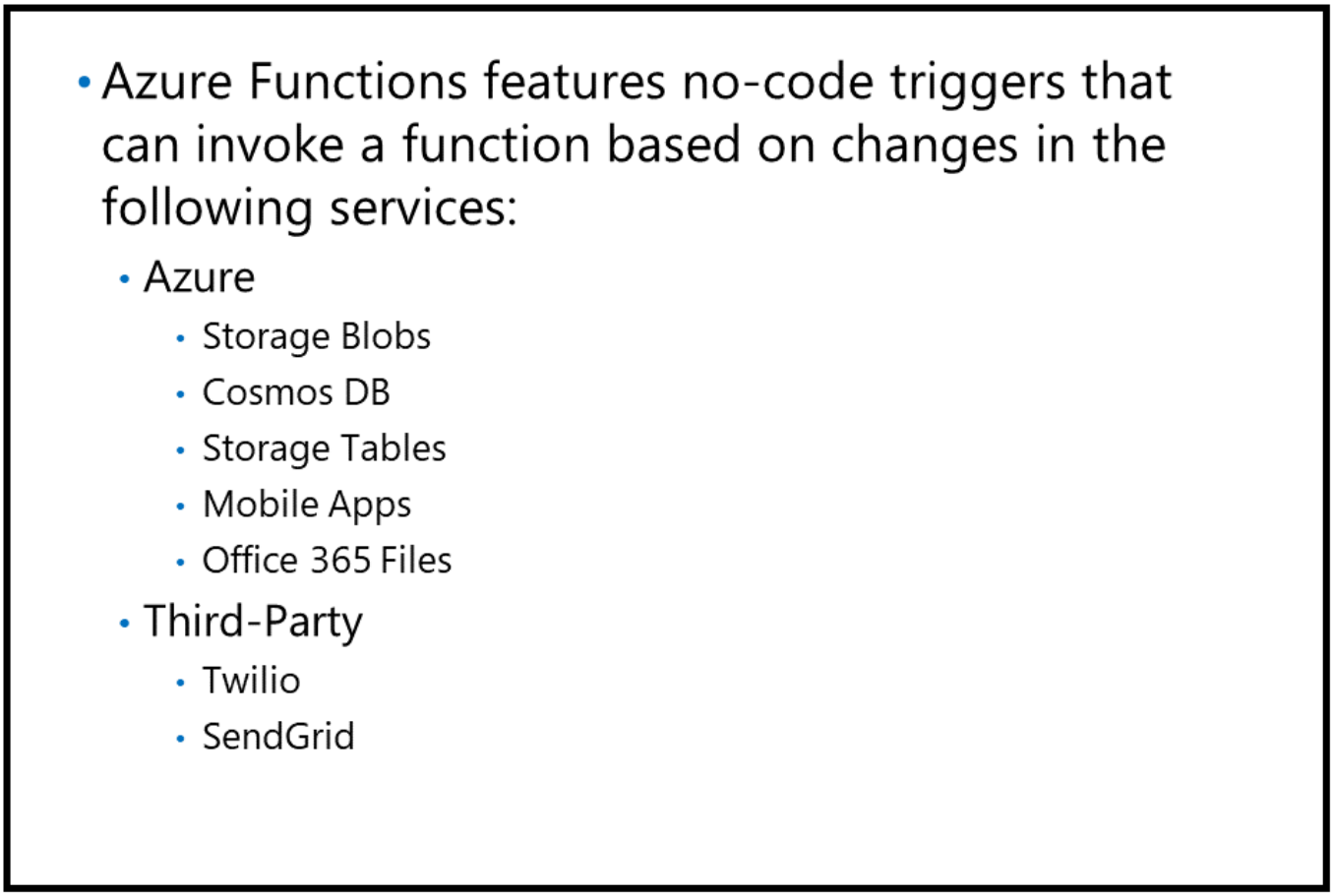
- **App Service**: The App Service model allows you to use the already familiar App Service Plan paradigm with Function Apps. In this model, Function Apps exist as simply another kind of app within an App Service Plan.
- **Consumption**: The consumption model allows you to pay for Function Apps based on execution time as opposed to having a dedicated App Service Plan. Function Apps are billed at a rate specific to Gigabyte Seconds after a specific free monthly allocation threshold has been met.

Function Apps share a lot of functionality with other App types such as Web Apps. You can configure App Settings, Connection Strings, AlwaysOn, Continous Deployment and many other settings that are configurable in a Web App

instance.

> **Note:** In order to guarantee that your Function App will respond to requests as quickly as possible, you should enable the AlwaysOn feature for your Function App whenever possible.

### Event-Based Triggers




Azure Functions must be initiated in some way before they can begin running code or processing data. A Trigger is anything that invokes an Azure Function.

Azure Functions have triggers for many different scenarios including:

- Creation of Blobs
- Changes to data in Cosmos DB
- External File or Table changes
- HTTP requests
- OneDrive or Excel files
- E-mail messages
- Mobile Apps
- SendGrid e-mails

- Twilio Text Messages

Additionally, Azure Functions can also be triggered by existing messaging services including:

- Service Bus
- Storage Queues
- Logic Apps
- Event Grid
- Event Hubs
- Webhooks

Azure Functions is unique in the sense that all of these triggers can be configured with minimal or no code. The developer team can focus on authoring the application

# Lesson 3: Integration

This lesson introduces API Management and Logic Apps as integration solutions available on the Azure platform.

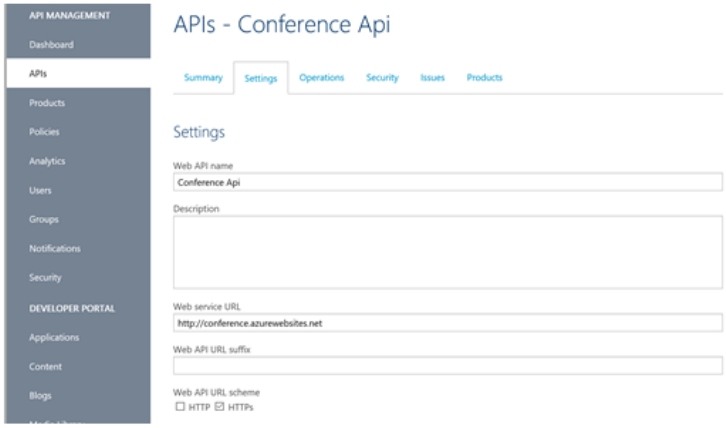
## Lesson objectives

After completing this lesson, you will be able to:

- Describe API Management in the context of a REST API B2B solution
- Use Logic Apps for sophisticated integration among application components.

## API Management

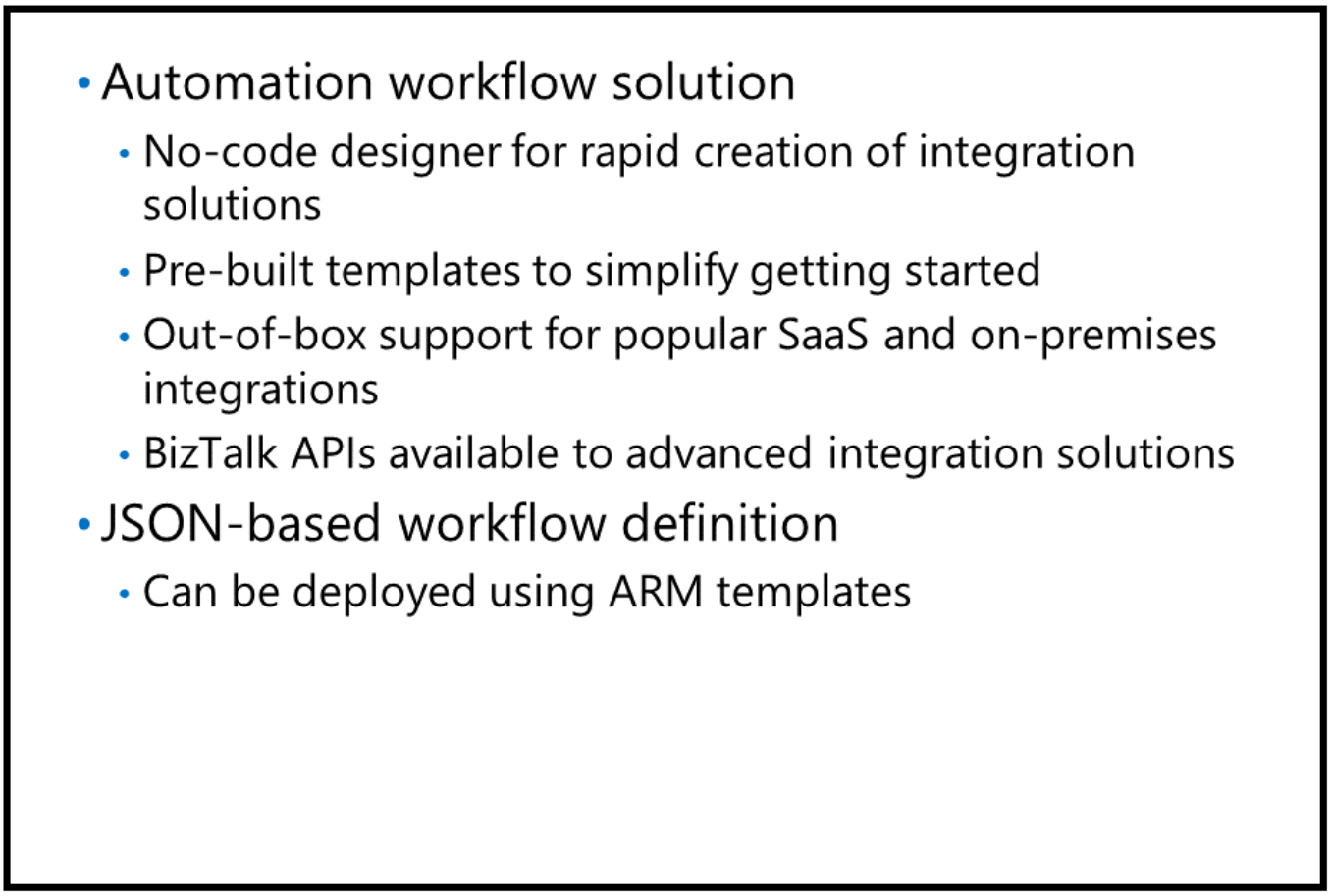
- API Proxy Service
  - Protect your API Endpoints
  - Add billing, throttling, monitoring and policies to existing APIs without changing their code
  - Manipulate requests and responses in-flight to meet business requirements




Azure API Management helps organizations publish APIs to external, partner and internal developers to unlock the potential of their data and services. Businesses everywhere are looking to extend their operations as a digital platform, creating new channels, finding new customers and driving deeper engagement with existing ones. API Management provides the core competencies to ensure a successful API program through developer engagement, business insights, analytics, security and protection.

To use API Management administrators create APIs. Each API consists of one or more operations, and each API can be added to one or more products. To use an API, developers subscribe to a product that contains that API, and then they can call the API's operation, subject to any usage policies that may be in effect.

## Logic Apps




Logic Apps are workflows, designed using JSON, that can connect various components of your application together using minimal or no-code. Logic Apps can integrate with existing services using built-in connectors. There are connectors available for popular services such as:

- SQL Server
- OneDrive
- Dropbox
- Twilio
- SendGrid
- Cosmos DB
- Event Grid
- Event Hubs
- Storage Blobs, Queues and Tables
- Mobile Apps

Logic Apps can also integrate with custom APIs that are deployed as API Apps in your subscription.

The main components of a Logic App are as follows:

- **Workflow**: The business process described as a series of steps, in an acyclic graph (loops are not supported)
- **Triggers**: The step that starts a new workflow instance
- **Actions**: A step in a workflow, typically a Connector or a custom API App
- **Connector**: A special case of API App ready made to integrate a specific service (e.g. Twitter) or data source (e.g. SQL Server)

## Lesson 4: High Performance Hosting

Azure provides the capability to create scalable and performance aware application using several PaaS Service together. Using a serverless model we can create global scale and high performance web applications.

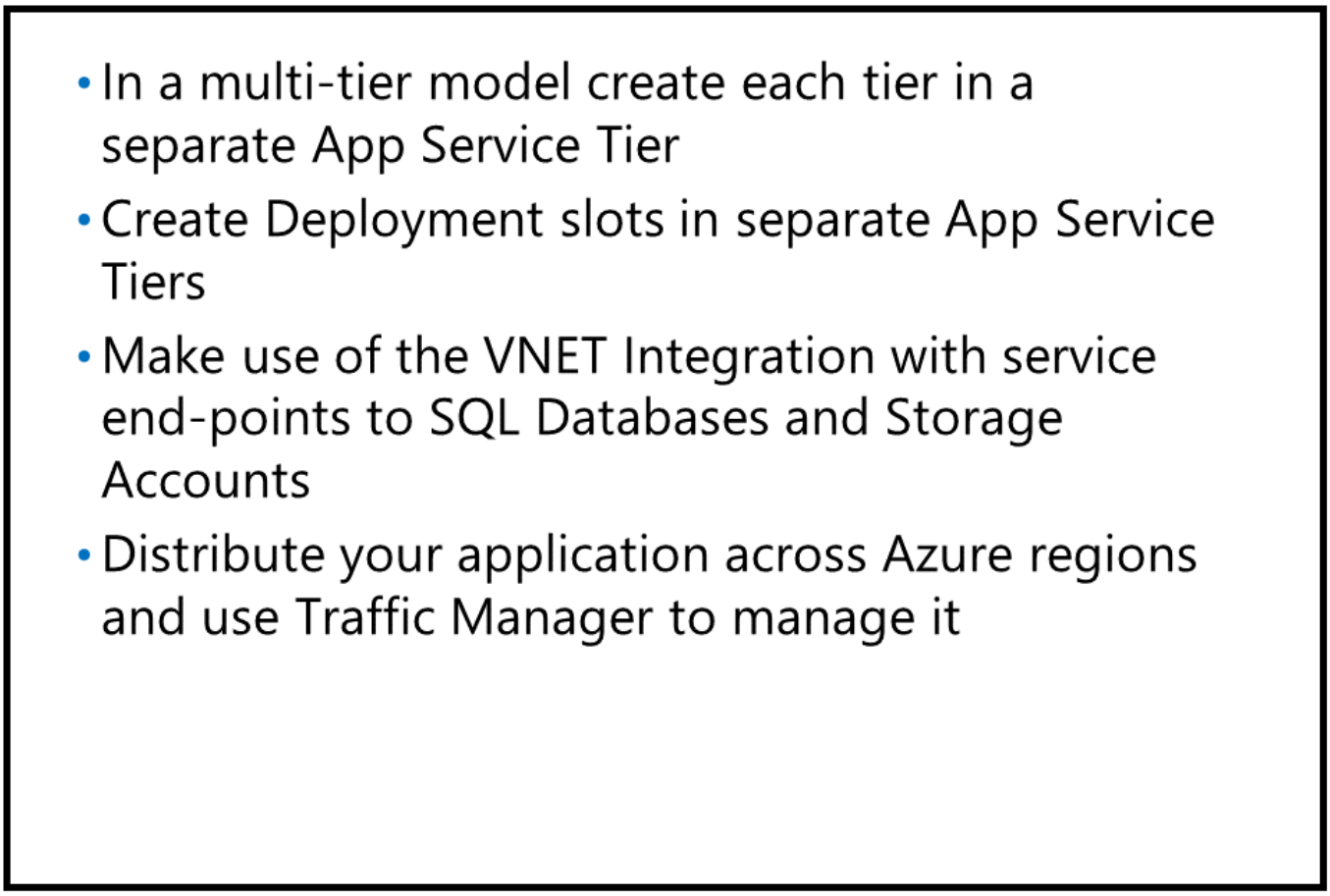
This lesson outlines how the App Service and related services can work together to create scalable and performance web applications.

### Lesson objectives

After completing this lesson, you will be able to:

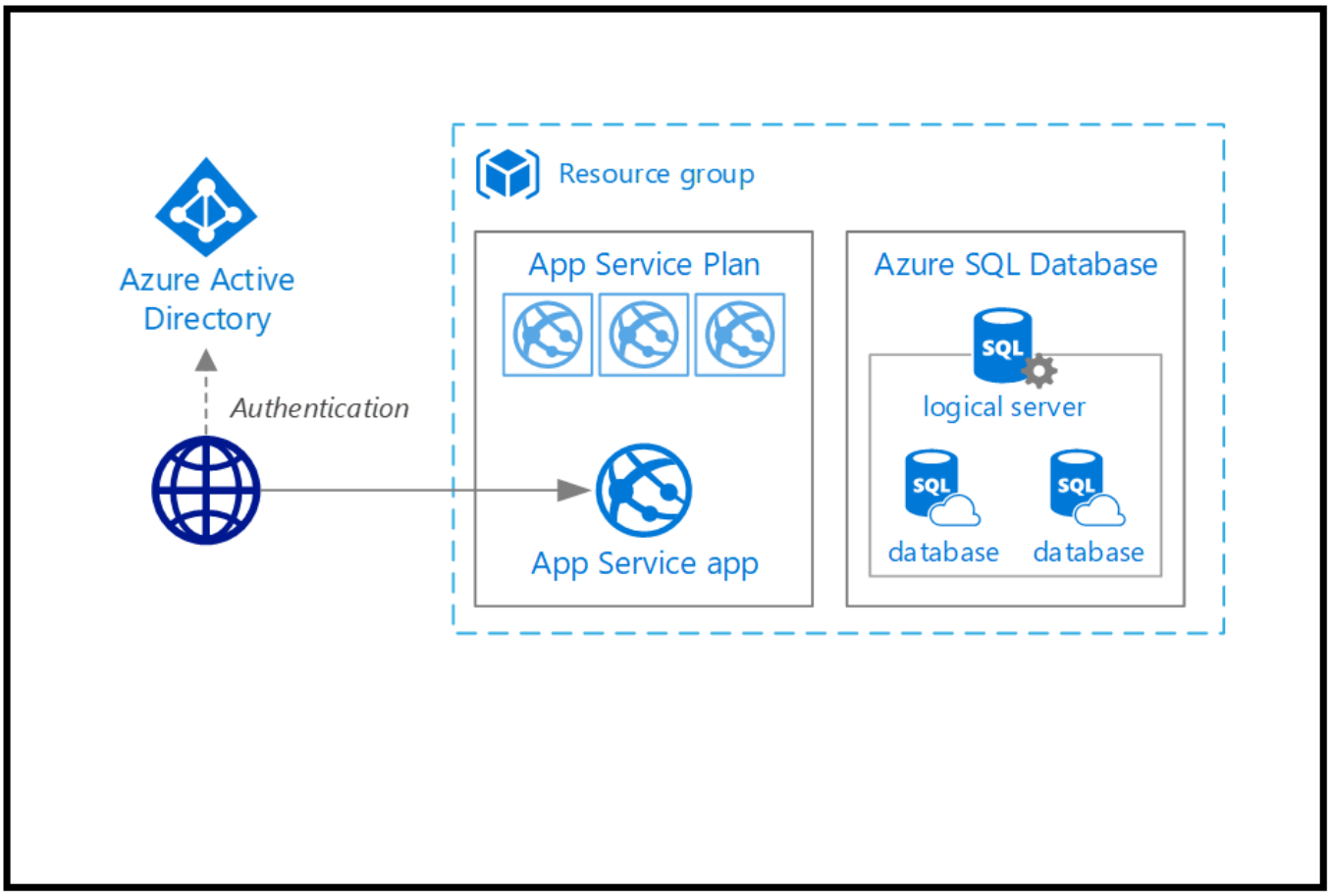
- Best Practices in creating performance aware web applications
- How to scale your web application
- Understand how multi-region apps can be built in Azure
- How to create a business continuity plan

### Best Practices




Azure allows you to create state of the art deployments for your web applications. Both Front-end and Back-end. Using the PaaS service App Service, the challenge of creating complex, global high-scalable solutions is simplified. Azure provides a set of services that will simplify the creation of such solutions. These services range from App Service, Traffic Manager, Redis Cache, Queues, Cosmo DB, ….

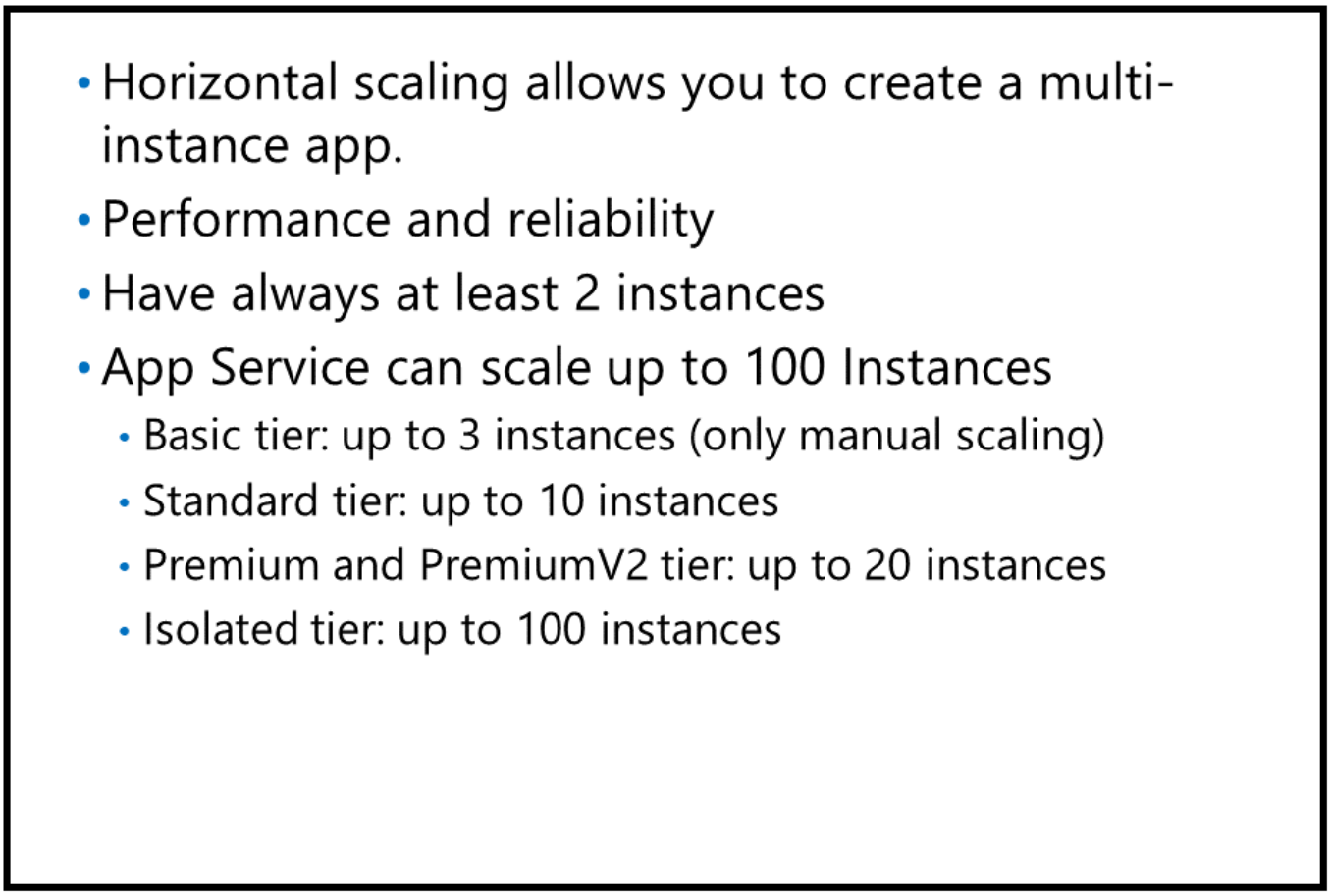
## Basic Web Application




In this architecture, we start by using the App Service that can have more than one application running in the same compute. We also have a logic SQL Server to store our data. It's recommended to use the Standard or Premium tiers in order to support autoscale and SSL.

It's also a recommendation to host both the app services as the SQL database in the same Azure region in order to minimize network latency.

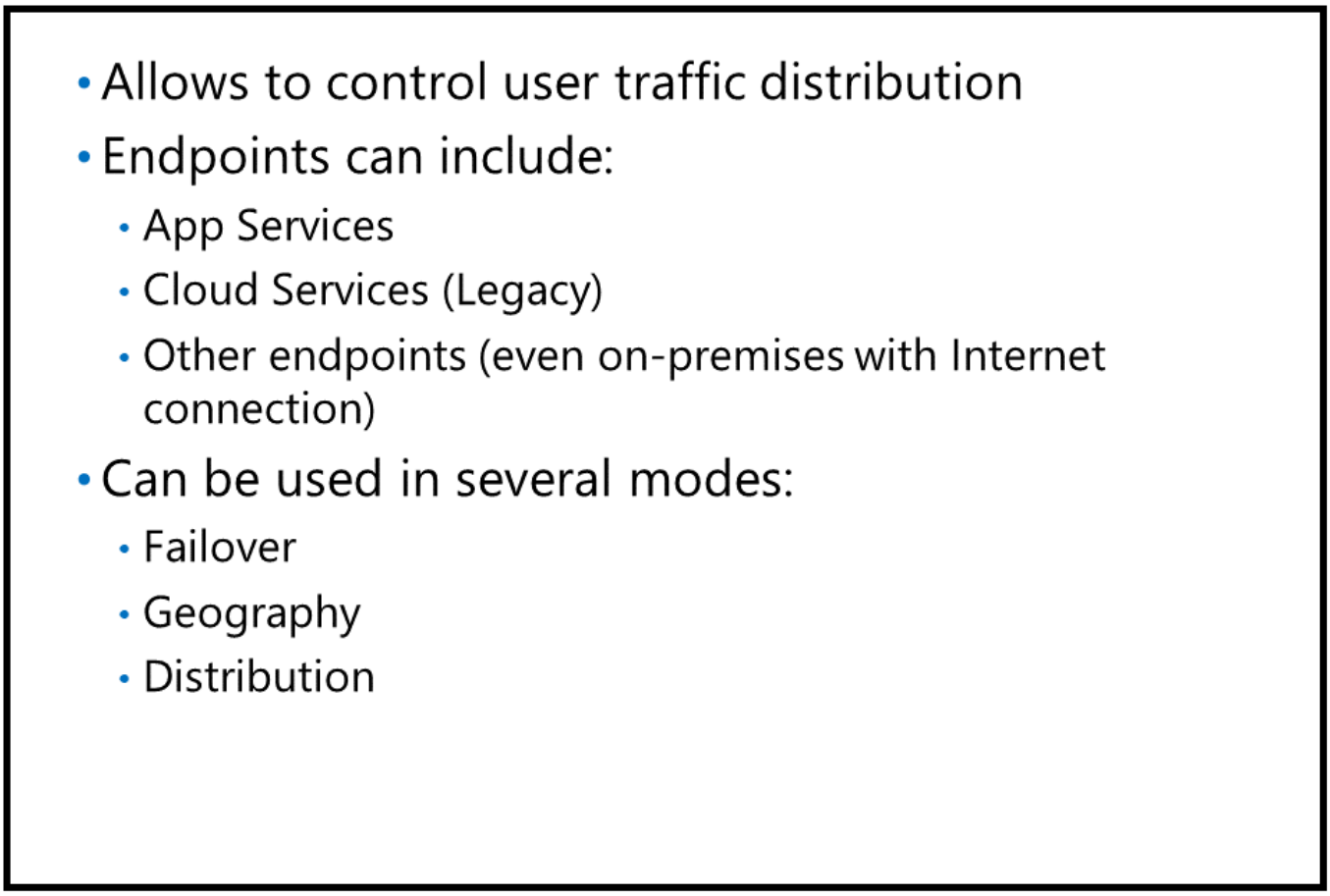
## Scaling




In Azure App Service, when an administrator needs to increase the performance he can do it by changing the number of instances running in the App Service Plan (scale out) or by changing to a higher pricing tier (scale up). Scaling up provides more CPU, memory, disk space and features, like autoscale, deployment slots, … Scaling out increases the number of instances running the application. This scaling can be manual, if you're using the Basic Tier, or automatically, if you're using the Standard or upper service tiers. Standard tier allows to scale up to 10 instances and if you still need more instances you can go to the Isolated tier where you can scale up to 100 instances. Scaling an application does not require you to change your application code.

In this architecture we had separate our two applications tiers (Web app and an REST API) into different App Service Plans. This way we can scale each tier individually. We had create a Web Job in order to process long-running task in the background, this is also accomplished by using a Message Queue to control the work to be done and already processed.

## Traffic Manager




Using a multi-region architecture provides a higher availability of your applications. If there is an outage in the primary datacenter, using the Traffic Manager service is possible to redirect the traffic to a secondary data center that is available, this way the application will not suffer an outage, reinforcing the overall application uptime.

In a multi-region model, we have 3 main components:

- Active and Standby regions
- Traffic Manager
- SQL Database Geo-replication

In a multi-region model, we have higher global availability of our solutions. This is accomplished by having our solution replicated in two regions and having the Traffic Manager routing incoming traffic to the active region, but in case of failure of this region, this service will failover to the standby region. The SQL Database active geo-replication feature also plays an essential role in this architecture. It creates a replica in the standby region and if the primary database fails or needs to be taken offline, we can failover to one of the up to four replicas.

## Lesson 5: Mobile Apps Case Study

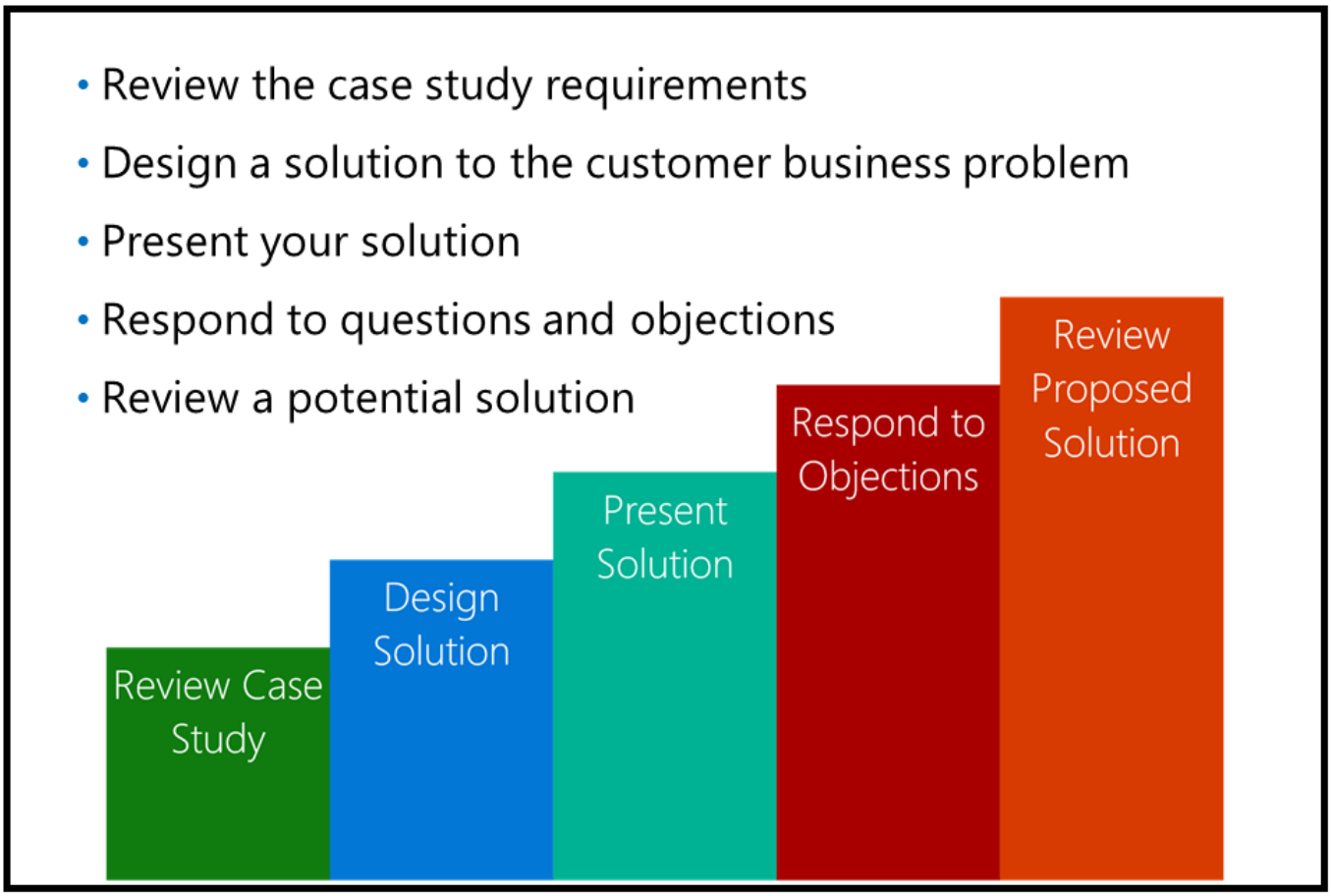
In this case study, we will look at a customer problem that requires an architectural recommendation.

### Lesson objectives

After this case study, you should:

- Identify customer problems as they are related to networking.
- Design a solution that will meet the customer's objectives.
- Ensure your designed solution accounts for customer objections.

## Case Study Overview




**Who Is the Customer?**

With Corporate Headquarters in Phoenix Arizona, the Crazy Taxi Cab Company has quickly risen to be the premier provider of private low-cost transportation in Arizona. Bucking industry norms, Crazy Taxi Cab Co drivers are company employees who work as a team, rather than independent contractors who essentially compete with each other for fares. The founding partners believed this would allow its drivers to focus on providing a great customer experience as opposed to simply "racing to the finish line". Crazy Taxi has developed a reputation for having fast, reliable, and friendly service, due largely in part to their extensive network of drivers, and their well-executed, albeit radio based dispatching approach.

Crazy Taxi is drowning in success. While dispatchers are reaching out to drivers to pick up customers who have called in, new callers often find themselves on hold, waiting for their calls to be answered. The executives at Crazy Taxi realize that they need to modernize their operation or risk losing business in the future. Their Chief Operating Officer Christopher Giovanni states that "while we function like a well-oiled machine, we're still running on 20th century equipment and we are already seeing signs that this is eroding our advantage over the competition…we need to bring our operation into the 21st century and that starts with FastRide".

**What Does the Customer Already Have?**

Crazy Taxi does not want to manage a more complex of a system than absolutely necessary. They already have three web and app developers, who have built their marketing pages as well as few proof of concept apps for iOS and Android, and one of them is quite savvy with .NET and brings some backend experience to the table. However, the executives all agree that this one developer cannot deliver the entire backend alone, so they are looking for a solution that provides them a “back end in a box” as they cannot afford to hire any more developers. Additionally, headquarters wants their IT team to have easy visibility into the health of the system, and more importantly that the system can take care of itself. Specifically, they are familiar with the notion of pay-as-you-go billing, and would love a solution that operates at nominal cost during their average daytime load, but on Friday night and Saturday night automatically handles the increased demand, albeit at an additional cost.

**What Is the Customer's Goal?**

FastRide is a software solution that the company has been planning to build that uses the GPS of 4G enabled mobile devices in each Crazy Taxi to monitor the location of each vehicle in the Crazy Taxi fleet. By being able to visualize how their fleet is moving throughout the city in real time, FastRide will allow Crazy Taxi Cab Co. to optimize their driver coverage throughout the city. When a new customer calls in, the telephone dispatcher enters their pickup location and the FastRide system identifies the closest available driver, instead of requiring dispatch to manually poll all of the drivers over the radio. While stationary, a driver accepts a fare using the FastRide app on his mobile device. FastRide will provide the driver with the pick-up details such as location, passenger name and callback phone (if available). Upon arrival the destination, the device will compute the fare automatically based on time in transit and distance driven, all with intent of minimizing driver distraction while driving due to interacting with the device. The fare information should be collected on the device and uploaded to a central database in near real-time. Should the driver enter a pocket without cellular connectivity, the device should store the data offline and sync it as soon as connectivity is restored.

The fare data is associated with each driver, who logs in in thru his or her app using his or her own Microsoft Account, Google, Twitter or Facebook credentials. It is the intent that driver fare data will be used to track the driver’s progress against established company goals, which both the company and driver access in the form of reports. Because these reports may grow to involve more complex computations (such as comparing the driver’s selected route to similar routes taken by other drivers), these data powering the reports will be computed nightly and made available the following day.

While initially notifications would be sent only to drivers about potential fares they can choose to accept, Crazy Taxi is looking forward to when they can offer their patrons a mobile app that can receive notifications about when the driver is in route, ETA updates and any messaging from the driver to the customer. They would like to be certain that the system they build on today has the capacity to meet those increased demands in the future.

**What Does the Customer Need?**

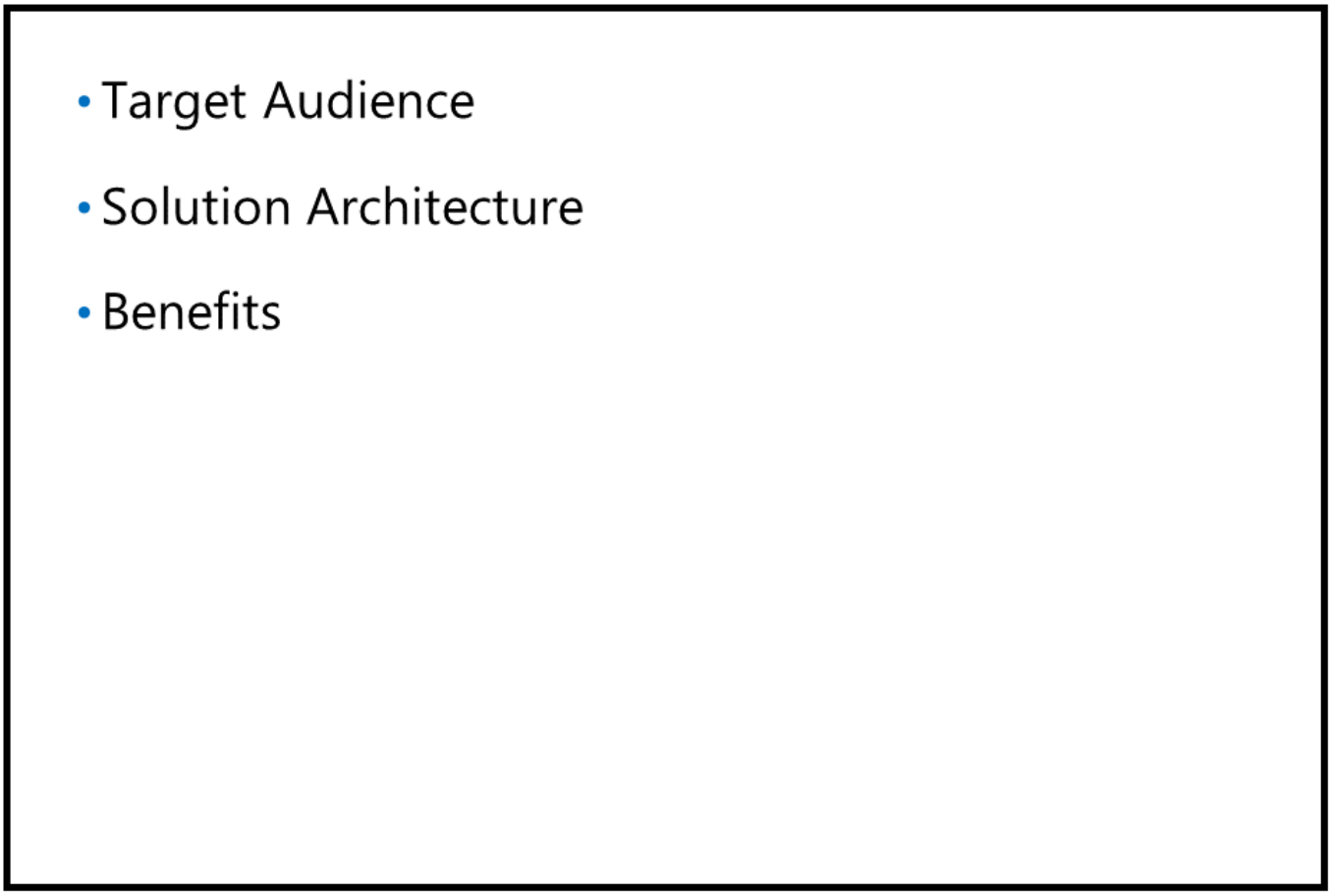
- A back-end as a service that provides both data storage, API support and push notifications.
- A solution that is quick to implement with a small team that has limited back-end development experience and capacity.
- A back-end solution that can be implemented using .NET.
- A solution that is easy to monitor for health and telemetry data.

**What Things Worry the Customer?**

- Doesn’t Azure Mobile Apps only work on Windows devices?

- Our development team doesn't know node.js. We had heard mention of Mobile Apps, but thought it only supported JavaScript backends.
- Our development team seems to think implementing push notifications using Apple and Android apps directly is easy, but we (as the executives) aren't so sure. How difficult can it be?
- Can't we just build all of our backend using Azure Web Apps?

## Case Study Solution




**Preferred Target Audience**

Operations Management Team led by Christopher Giovanni, Chief Operating Officer at Crazy Taxi Cab Co.

**Preferred Solution**

Crazy Taxi Cab Co. liked the idea of a highly scalable, agile solution that they could both execute on and manage with a small IT team with limited back-end development experience in .NET. The company's Microsoft representative introduced them to Azure Mobile Apps, a cloud based mobile platform that provides backend CRUD data services, offline data sync for mobile devices, custom REST API services, push notifications and services for login via a social identity provider. This was exactly the "back-end in a box" that they were looking for.

Crazy Taxi Cab Co. leveraged the many features of Mobile Apps available to them with the Standard tier in order to minimize their backend development burden and accelerate solution delivery.

The implementation of the proposed Azure Mobile Apps solution would create a strategic partnership that will help Crazy Taxi Cab Co. to overcome its challenges with:

- Minimizing system downtime
- Sending multiple user specific notifications to mobile devices
- Managing user accounts leveraging social identity providers, such as Microsoft Account, Facebook, and Twitter
- 24/7 (secure) data accessibility throughout the Crazy Taxi Cab Co. network
- Scalability in software solutions in an agile marketplace

As the FastRide system continues to be improved upon, Mobile Apps will help to inject velocity into the development cycle by providing a mobile back end to the application. Mobile Apps offers cross platform compatible components, which gives Crazy Taxi Cab Co. the flexibility to change their mobile platform as the needs of their business dictate. This “back-end as a service” approach will allow Crazy Taxi to focus on building an app that merges the right functionality with a great user experience for each market the operate in.

By utilizing push notifications, Crazy Taxi Cab Co. can optimize their customer pickup messages through the FastRide app. This allows for faster, and more streamlined in-app communication. When a new fare’s pickup address is entered into FastRide by a dispatcher, or the customer facing mobile app, Mobile Apps will enable FastRide to automatically send a notification to the closest available driver—eliminating the need for manual notifications. Since push notifications can be managed, each base will have control over the messages sent to its drivers.

To propose a more complete solution and ensure deployment success, it would be helpful to know:

- Type and operating system of tablets being used
- Expected product life of current tablet choice
- Current number of dashboard tablets
- Projected number of tablets after the planned expansion this year and next year
- Current average number of fares completed per day, week, month, year
- Projected average number of fares completed per day, week, month, year after the expansion into new markets
- Rate of growth across bases (customer and driver)
- Other software products used to operate the company

Understanding these details and decisions will help identify the current and future software, hardware, and infrastructure needs of Crazy Taxi Cab Co., and to provide solutions that are consistent with their short and long-term business goals.

**HIGH-LEVEL ARCHITECTURE**

Crazy Taxi Cab Co. leveraged the many features of Mobile Apps available to them with the Standard tier in order to minimize their backend development burden and accelerate solution delivery.

**Authentication**

Drivers login to the FastRide app on their device using their Microsoft, Google, Twitter or Facebook credentials, the federated login process being handled by Mobile Apps in conjunction with the aforementioned identity providers.

**Notifications**

Once logged in, the app registers with Mobile Apps, associating the driver's identity with the Notification Hub (associated with the Mobile App). In this way, Crazy Taxi dispatch can send broadcast notifications to all drivers, but still be able to send targeted Fare Alert messages to a particular driver

By having Mobile Apps in place with Push Notifications, Crazy Taxi Cab Co. is well positioned in the future to light up the ability to deliver a customer-oriented app that deliver push notifications to customers informing them of events relevant to their pickup.

**Offline Data**

For the driver's iOS and Android devices, in the construction of the FastRide app, they leveraged the native client Offline Data Sync functionality for synchronizing Fare Data when temporarily disconnected from the cellular network. This Fare Data is stored using Tables in Mobile Apps, which ultimately rests as relational tables in a SQL Database. This SQL Database also stores the driver profiles (that associate social credentials with driver profile information).

**Back End Custom Services**

When a driver accepts or declines a fare received via a push notification, that message is sent using a Custom REST API hosted by Mobile Apps and built using ASP.NET Web API.

**Front-End Website**

Dispatch uses a website hosted in Azure Websites to manage the taxi cab dispatch process. The Notification Hub is configured so that only the dispatch website is allowed to send push notifications to the drivers (the FastRide app for drivers is Listen only).

**Monitoring**

Crazy Taxi corporate IT monitors the health of the solution using the Dashboard for the Mobile App or Website in the Azure Portal. To assist the IT team with visibility into the health of the system, they should configure monitoring endpoints, again using the Azure Portal, on their website and Mobile Apps and enable e-mail alerts should the Response Time and Uptime metrics for those fall below values they deem acceptable.

**Scaling Configuration**

They have configured Autoscale on their Mobile App, via the Scale tab in the portal, to double the number of instances it uses on Friday and Saturday night in order to handle the increased load, then return back to their normal instance count for the rest of the week.

**Backend Jobs**

They have also created a Mobile Apps Scheduled Job that processes the Fare Data on nightly basis to generate the data sets that power the Fare Reports. This data is stored in the same SQL Database that stores all the other data used by the solution.

Preferred Solution Diagram

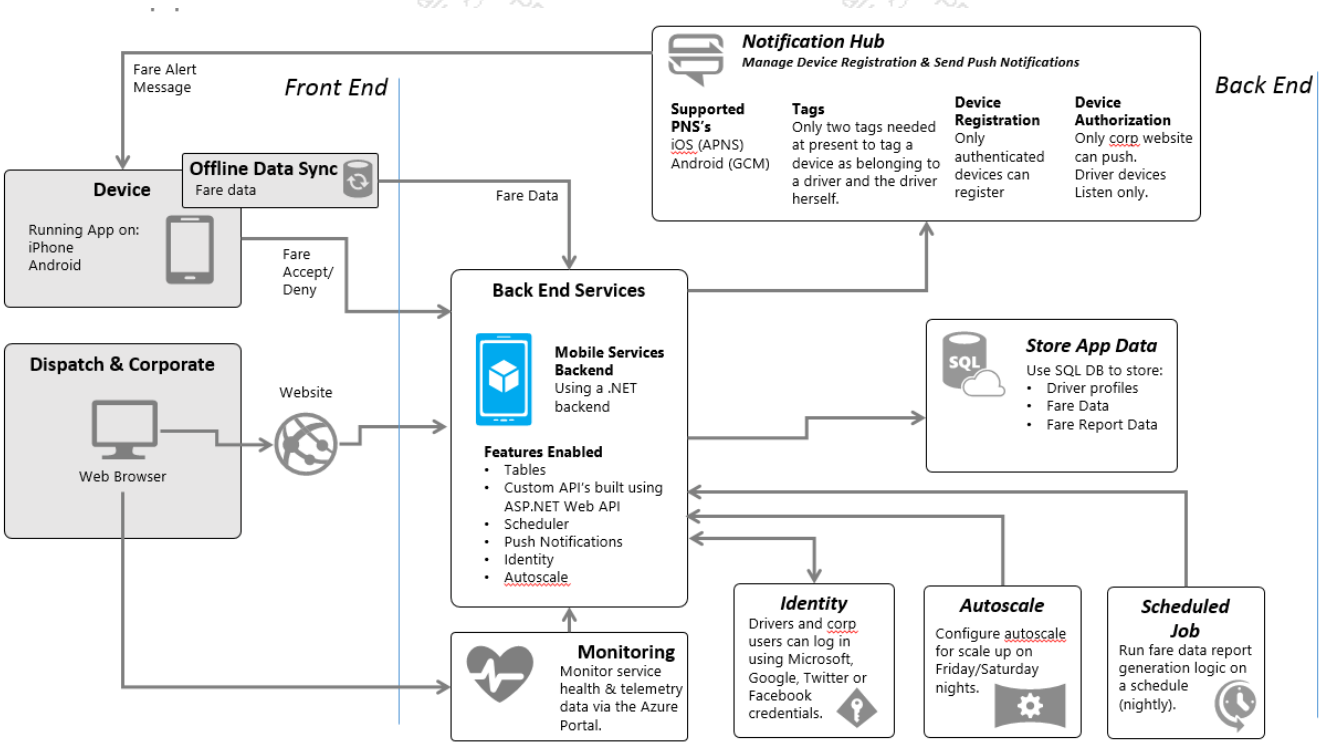



**FIGURE 5.1: DIAGRAM SHOWING THE PREFERRED SOLUTION USING MOBILE APPS**

**CHECKLIST OF PREFERRED OBJECTION HANDLED**

**Doesn't Azure Mobile Apps only work on Windows devices?**

Azure Mobile Apps provides native clients for iOS, Android, Xamarin, PhoneGap, Sencha and Appcelerator in addition to Windows universal C#, Windows universal JavaScript and Windows Phone. In addition, Azure platform services offer REST APIs that extend the reach to platforms for which there is not a native API, but are capable of making REST style requests.

**Our development team doesn't know node.js. We had heard mention of Mobile Apps, but thought it only supported JavaScript backends.**

Mobiles Services supports using .NET for the backend logic, and node.js (or JavaScript logic) does not have to be used anywhere in the backend code.

**Our development team seems to think implementing push notifications using Apple and Android apps directly is easy, but we (as the executives) aren't so sure. How difficult can it be?**

While using the Push Notification System of a particular device platform directly is typically made fairly simple by the provider of that platform (e.g., iOS apps have a straightforward API for leveraging Apple's Push Notification System), this simplicity is only true for the individual device and does not address the complete solution that requires at minimum a backend managing device registrations at scale and sending push notifications cross platforms in a timely fashion. Azure Mobile Apps provides that backend functionality, which can be easily scaled to meet demand.

**Can’t we just build all of our backend using Azure Web Apps?**

Azure Web Apps is effectively a superset of Mobile Apps and so can be used to implement the backend for Mobile Applications. However, Web Apps do not deliver the services tailored for the mobile application scenario, requiring the developers to write their own logic to integrate with Notification Hubs, SQL Database, Identity services and WebJobs. Additionally, Mobile Apps is prescriptive in the patterns used for developing custom API’s, and so speeds the development of such API’s by allowing the development efforts to focus on the business logic instead of dealing with structural and hosting decisions. These become important factors to consider when taking into account the development team size, capabilities and timeframe.

**PROOF OF CONCEPT CHECKLIST**

The primary items a Proof of Concept for this solution could demonstrate include:

- Scalability / Scheduled Autoscale
- Mobile Apps ease of integration (e.g., the backend in a box)
- Streamlined communication with Push notifications
- Integration of social identity platforms to aid in customer authentication and profile management
- Device offline data storage & synchronization
- Monitoring solution health

The Proof of Concept will be considered successful if the Crazy Taxi Operations Management Team believes they can realize value in:

- Speeding up the delivery of the overall solution
- Push notifications to streamline communication and send fare updates to tablet devices
- Authenticating users via social media platforms and future benefits of successfully leveraging social media integration.
- Minimizing system downtime by keeping app data in the cloud
- Scalability in a mobile cloud solution

The personnel resources you would leverage to build the PoC, may include:

- Partner Resources in the Region or MCS to help assist with migration design and implementation
- Microsoft Azure CAT for level 300 expertise requests with Azure

## Lab: Deploying Serverless Workloads to Azure

### Scenario

A startup company has hired your organization to help them take their web application, stored in a Git repository, and deploy the application and its API to Azure.

### Objectives

- Create Web App using an ARM Template.
- Deploy application code to existing Web App using ARM Template deployment resources.
- Deploy a Function App and associated code using an ARM Template.

***Lab setup***

Estimated Time: 90 minutes

Virtual machine: **20535A-SEA-ARCH**

User name: **Admin**

Password: **Pa55w.rd**

The lab steps for this course change frequently due to updates to Microsoft Azure. Microsoft Learning updates the lab steps frequently, so they are not available in this manual. Your instructor will provide you with the lab documentation.

### Exercise 1: Create Web App

### Exercise 2: Deploy Web App Code

### Exercise 3: Deploy Function App and Code

### Exercise 4: Cleanup Subscription

### Review Question(s)

## Module review and takeaways

### Review Question(s)