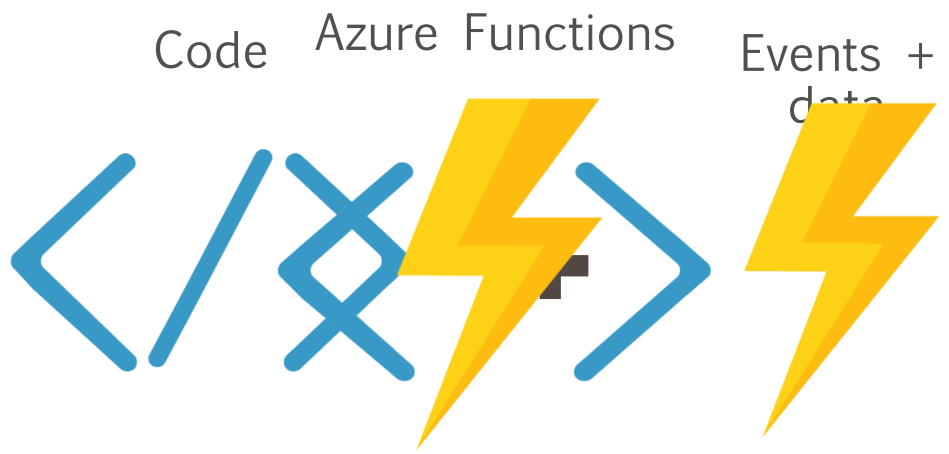
SERVERLESS PROGRAMMING USING AZURE FUNCTIONS

Sonu Sathyadas

Azure Functions

- Serverless compute service
- Run code on-demand without having to explicitly provision or manage infrastructure.
- Run a script or piece of code in response to a variety of events.
- Choice of language - C#, F#, Node.js, Python, or PHP
- Pay only for the time your code runs
- Azure is responsible for scaling the app as needed.

Azure Functions



Process events with Serverless code.

Make composing Cloud Apps insanely easy

Develop Functions in C#, Node.js, F#, Python, PHP, Batch and more

Easily schedule event-driven tasks across services

Expose Functions as HTTP API endpoints

Scale Functions based on customer demand

Easily integrate with Logic Apps

Features of Azure Functions

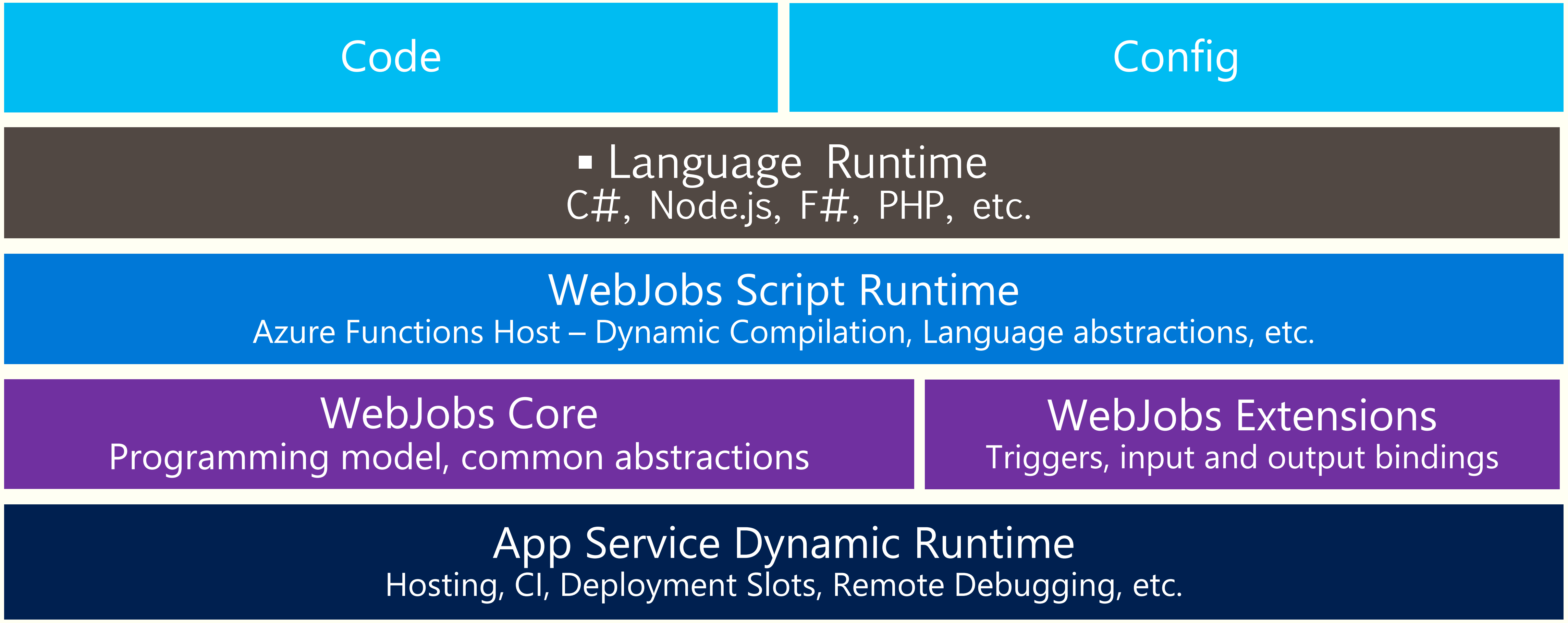
- Choice of language
- Pay-per-use pricing model
- Bring your own dependencies - Functions supports NuGet and NPM
- Integrated security
- Simplified integration with SaaS solutions
- Flexible development – Code on portal or Continues Integration through other sources/tools.
- Open-source – Runtime available on GitHub

What to do with “Functions”?

- Develop IoT solutions
- Processing data
- Integrations systems
- Simple APIs
- Microservices
- Scheduled tasks

Azure Functions architecture

Built on top of App Service and WebJobs SDK


Code
Config
▪ Language Runtime
C#, Node.js, F#, PHP, etc.
WebJobs Script Runtime
Azure Functions Host – Dynamic Compilation, Language abstractions, etc.
WebJobs Core
Programming model, common abstractions
WebJobs Extensions
Triggers, input and output bindings
App Service Dynamic Runtime
Hosting, CI, Deployment Slots, Remote Debugging, etc.

Service Integrations

- Azure Functions integrates with various Azure and 3rd-party services.
- These services can trigger your function and start execution, or they can serve as input and output for your code
- The following service integrations are supported by Azure Functions.
  - Azure Cosmos DB
  - Azure Event Hubs
  - Azure Mobile Apps (tables)
  - Azure Notification Hubs
  - Azure Service Bus (queues and topics)
  - Azure Storage (blob, queues, and tables)
  - GitHub (webhooks)
  - On-premises (using Service Bus)
  - Twilio (SMS messages)

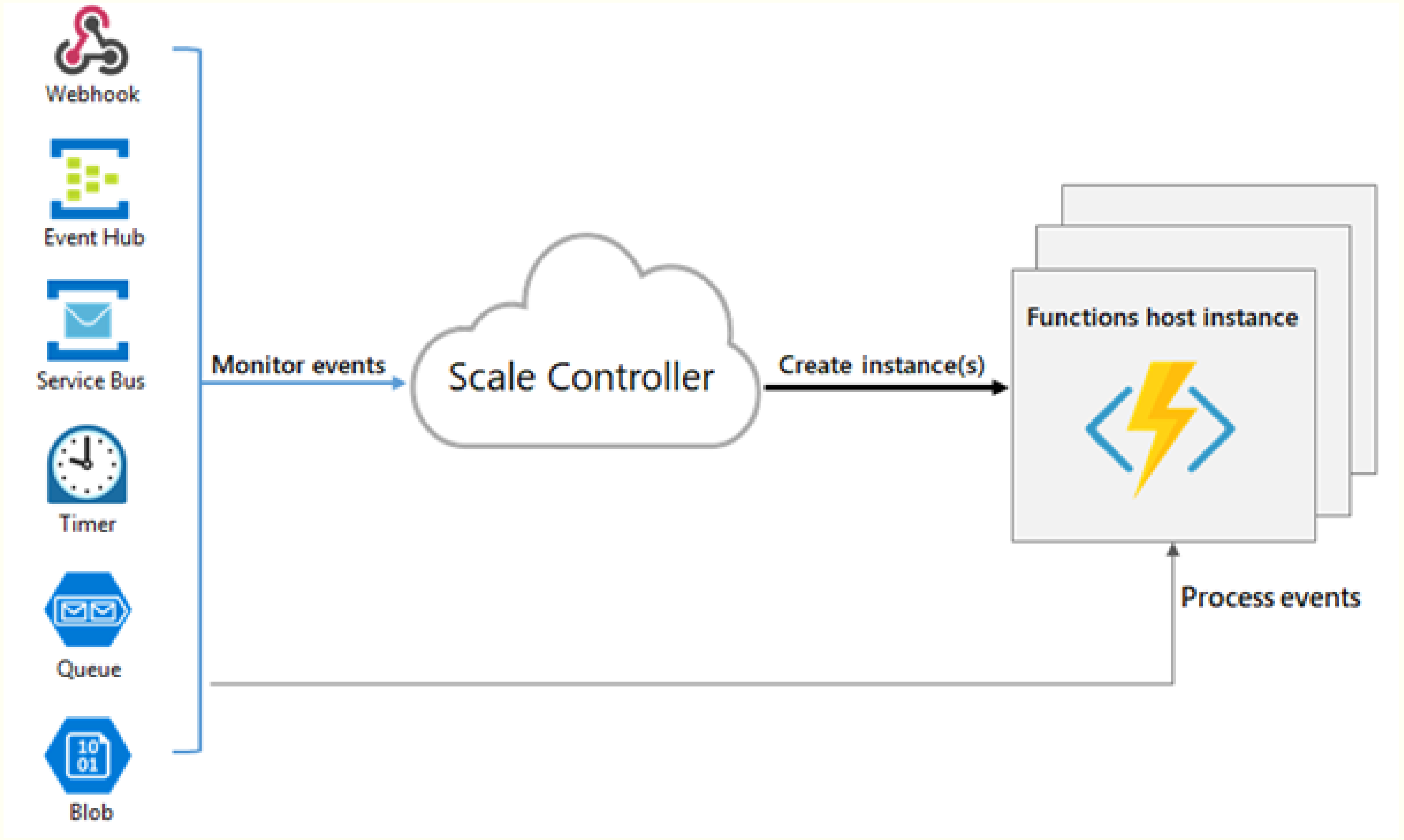
Pricing and scaling

- Azure Functions has two kinds of pricing plans
  - Consumption Plan
  - App Service plan tiers
- Choose the hosting plan during the creation of the function app.
- Hosting plan can't changed afterward.

Azure Functions hosting - Consumption plan

- Azure provides computational resources
- Pay only for the time code runs
- Platform responsible for scale
- A function can run for a maximum of 10 minutes (default timeout is 5 minutes).
- Billing is based on number of executions, execution time, and memory used.
- Each instance of Function host is limited to 1.5GB of memory.

Consumption plan scaling



Consumption plan billing model

- The following are units for billing:
  - **Resource consumption in gigabyte-seconds (GB-s).**

    memory size X execution time
  - **Executions.**

    Counted each time a function is executed in response to an event trigger.

Azure Functions hosting - App Service Plan

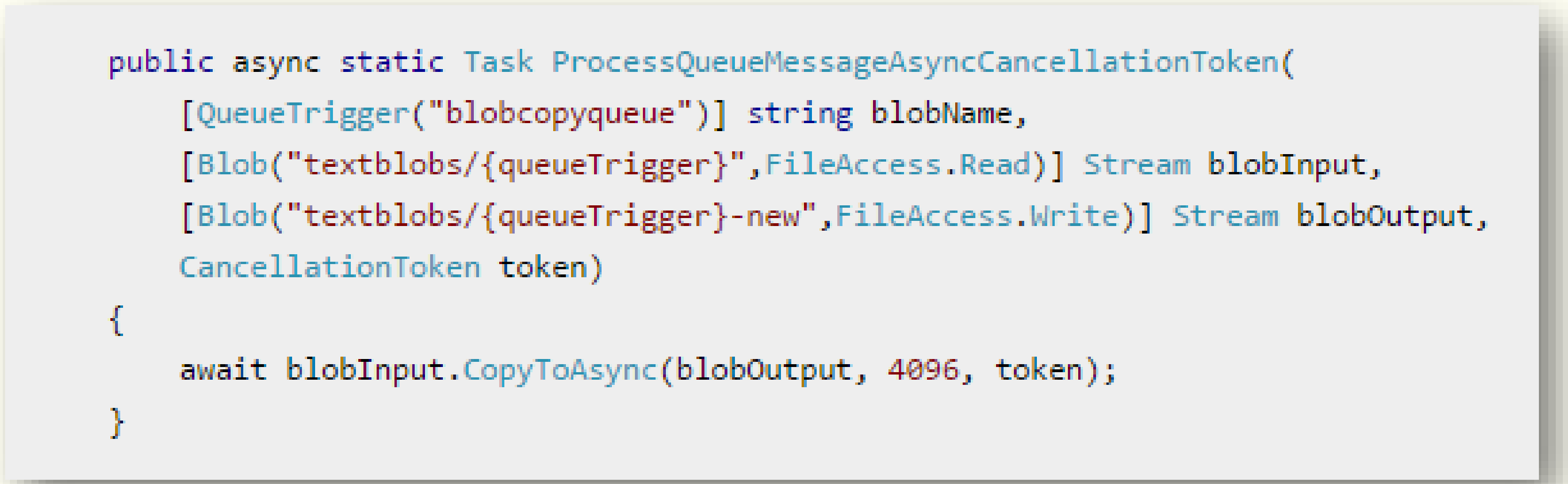
- Use existing App Service Plans
- Runs same as Web, Mobile and API apps
- Run on dedicated VMs on Basic, Standard, Premium, and Isolated SKUs
- No additional cost
- User is responsible for scaling (can enable autoscale).
- ‘Always On’ must be enabled.

Storage account requirement

- Both Consumption plan and App Service plan requires a storage account
- Storage account must support blob, queue, table and files.
- Blob-only and premium storage accounts NOT supported.
- Storage accounts are used internally for managing triggers and logging function executions.
- Function code files are stored on Azure Files shares of SA.

What is the "Functions" programming model?

- Function as the unit of work
- Functions are executed; they start and finish
- Functions have inputs and outputs



```
public async static Task ProcessQueueMessageAsyncCancellationToken(
    [QueueTrigger("blobcopyqueue")] string blobName,
    [Blob("textblobs/{queueTrigger}",FileAccess.Read)] Stream blobInput,
    [Blob("textblobs/{queueTrigger}-new",FileAccess.Write)] Stream blobOutput,
    CancellationToken token)
{
    await blobInput.CopyToAsync(blobOutput, 4096, token);
}
```

Best practices for the “Functions” model

- Functions *should* “do one thing”
- Functions *should* be idempotent
- Functions *should* finish as quickly as possible