Microsoft

Azure Functions 2.0: Enterprise-Grade Serverless

Minnesota Developers Conference
October 3, 2018

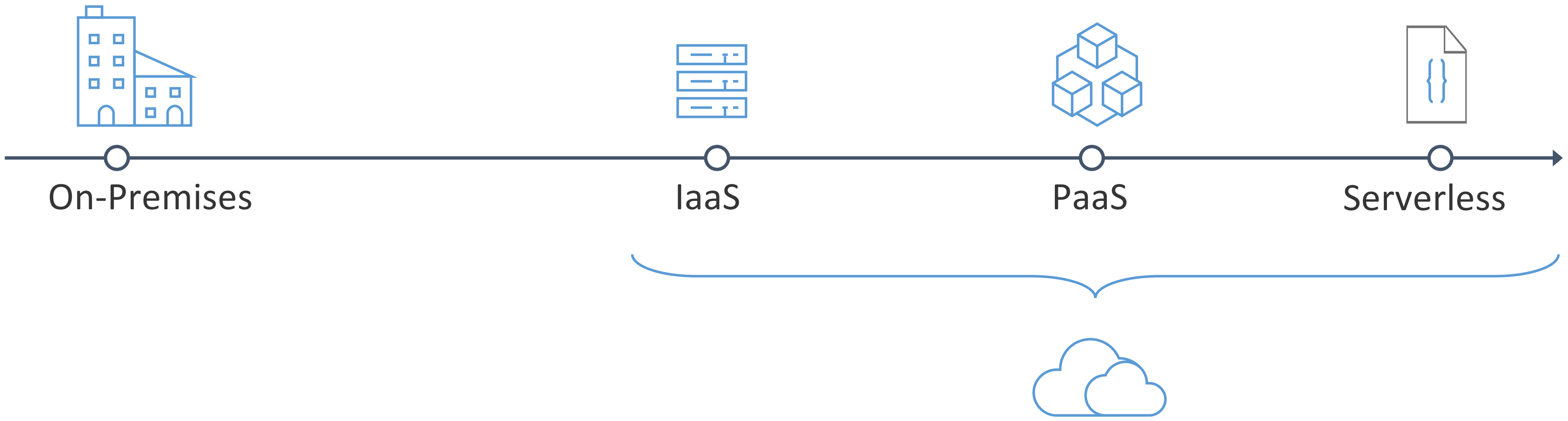
Katy Shimizu

Katy Shimizu

Software Engineer II, Azure Functions

@kashimizMSFT
katy.shimizu@microsoft.com

The “Evolution” of Application Platforms



What is serverless?

Abstraction of servers

Event-driven/ instant scale

Micro-billing

What are the benefits?

Focus

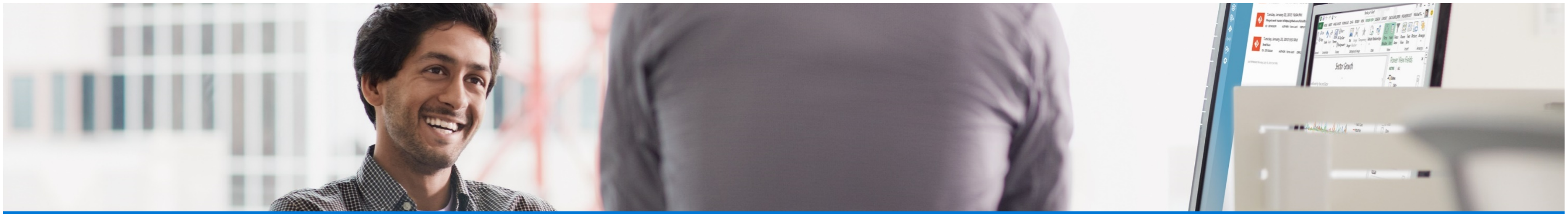
Solve business problems—not technology problems related to undifferentiated heavy lifting

Efficiency

Shorter time to market
Fixed costs converted to variable costs
Better service stability
Better development and testing management
Less waste

Flexibility

Simplified starting experience
Easier pivoting means more flexibility
Easier experimentation
Scale at your pace—don’t bet the farm on Day 1
Natural fit for microservices

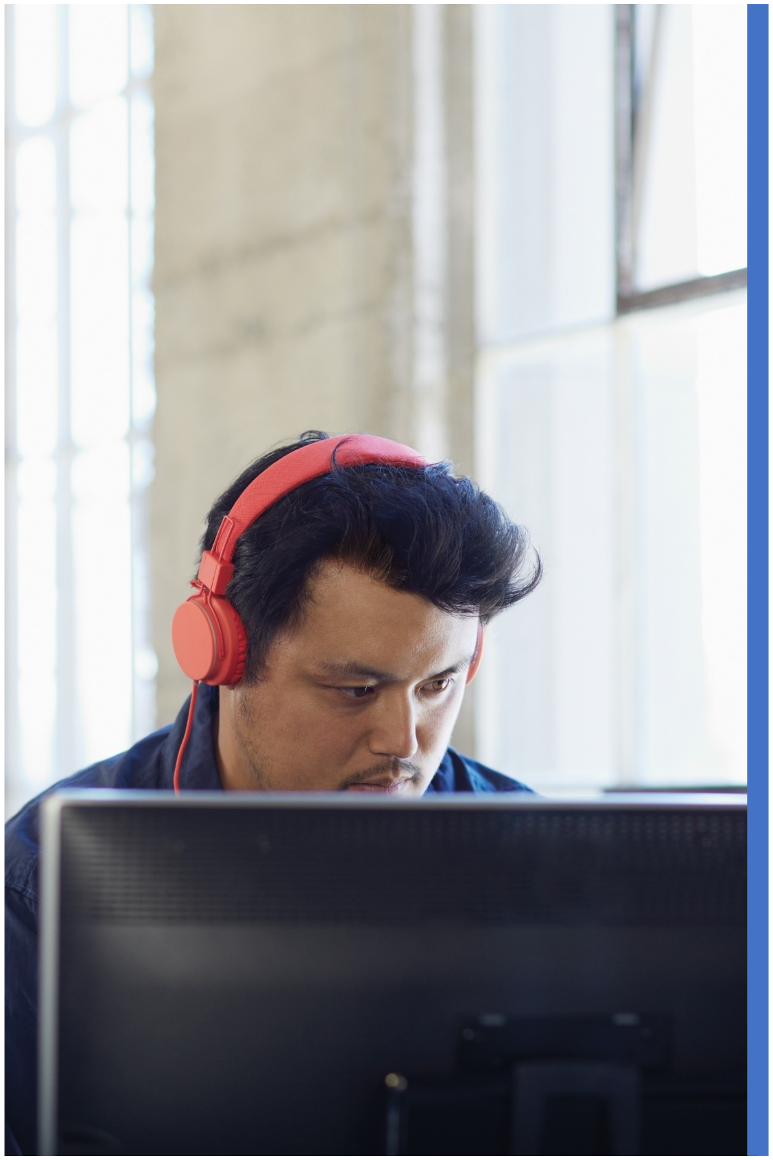


Focus on code, not plumbing

No infrastructure
management

Auto-scale based
on your workload

No wasted resources,
pay only for what you use



Sample scenarios for Functions

Web/Mobile app workloads

IoT-connected backends

Real-time processing

Automation of infrastructure

Full integration with Azure ecosystem

Development

- IDE support
- Integrated DevOps
- Local development
- Monitoring
- Visual debug history

Platform

Event Grid

Manage all events that can trigger code or logic

Functions

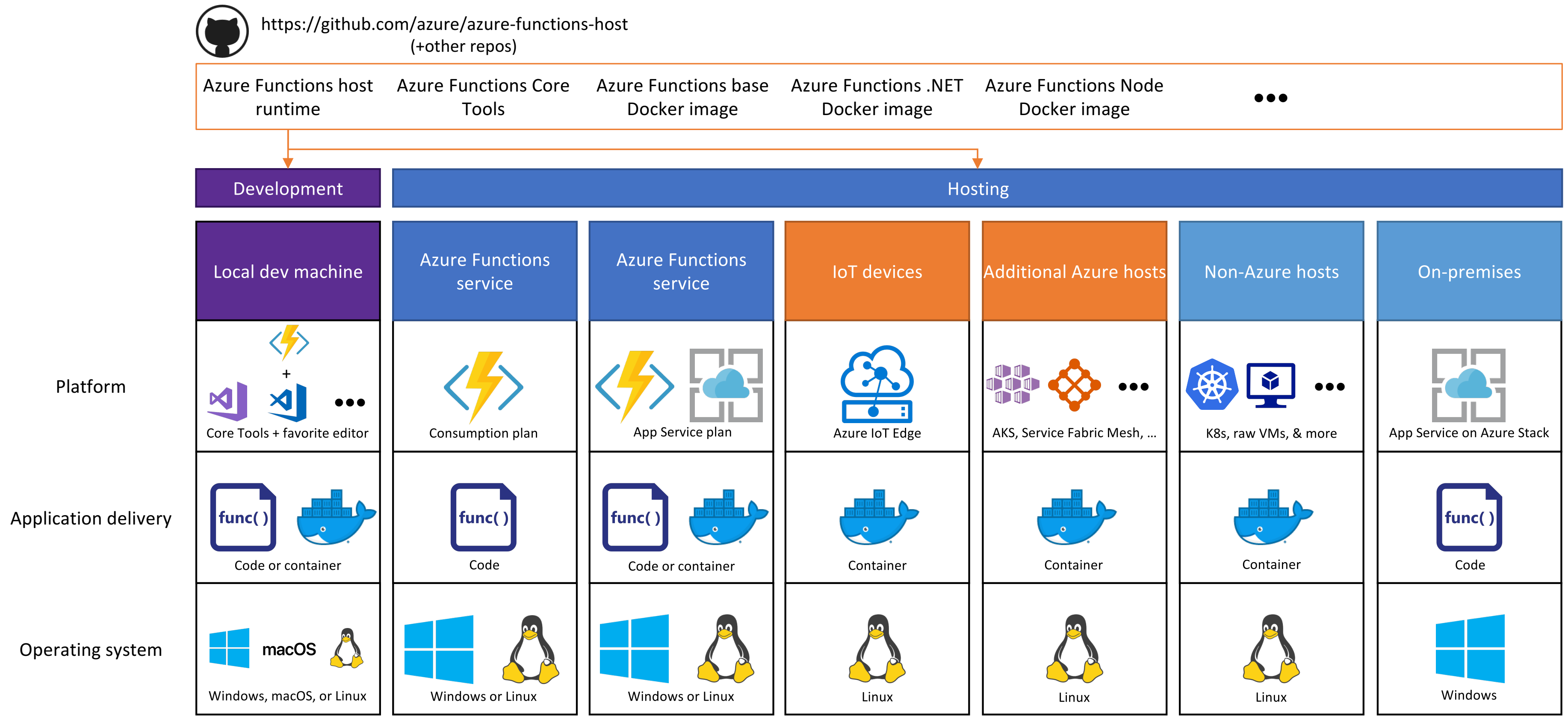
Execute your code based on events you specify

Logic Apps

Design workflows and orchestrate processes

- Database
- Storage
- Analytics
- Intelligence
- Security
- IoT

Functions everywhere



https://github.com/azure/azure-functions-host
(+other repos)

Azure Functions host runtime | Azure Functions Core Tools | Azure Functions base Docker image | Azure Functions .NET Docker image | Azure Functions Node Docker image | •••

	Development	Hosting					
	Local dev machine	Azure Functions service	Azure Functions service	IoT devices	Additional Azure hosts	Non-Azure hosts	On-premises
Platform	Core Tools + favorite editor	Consumption plan	App Service plan	Azure IoT Edge	AKS, Service Fabric Mesh, …	K8s, raw VMs, & more	App Service on Azure Stack
Application delivery	Code or container	Code	Code or container	Container	Container	Container	Code
Operating system	Windows, macOS, or Linux	Windows or Linux	Windows or Linux	Linux	Linux	Linux	Windows

Language options

Generally available

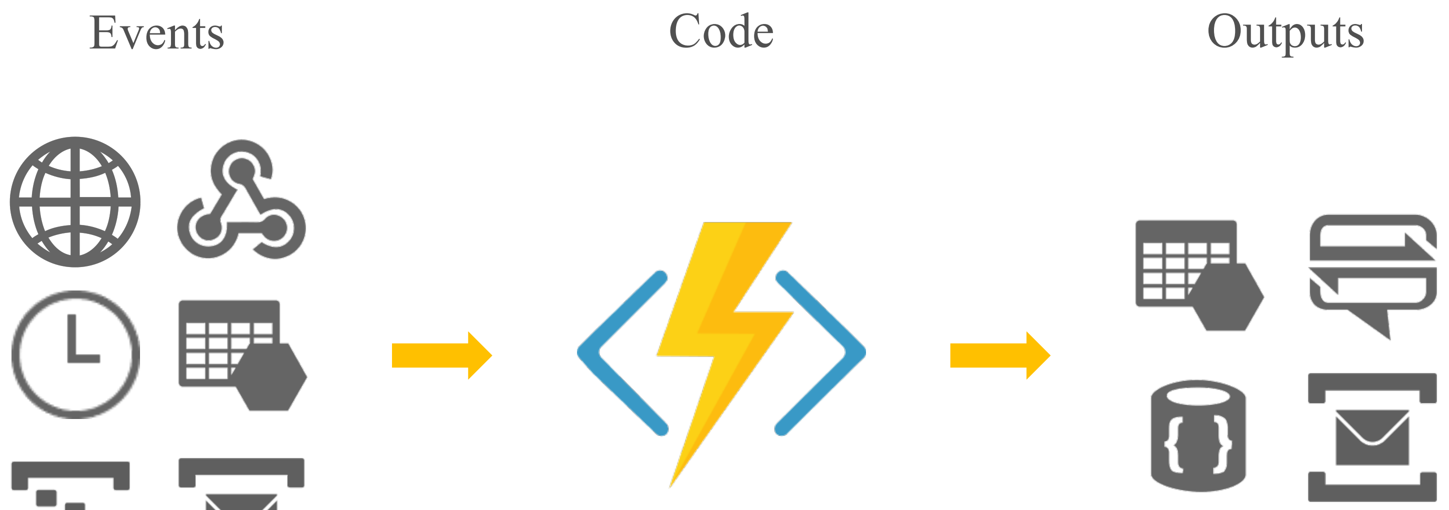
Public preview

Private preview

New!

More on the way!

Azure Functions



React to timers, HTTP, or events from your favorite Azure services, with more on the way

Author functions in C#, F#, Node.JS, **Java**, and more

Send results to an ever-growing collection of services

Bindings and integrations

Functions 1.0

Microsoft.NET.Sdk.Functions (.NET Framework 4.6)

- HTTP
- Timer
- Storage
- Service Bus
- EventHubs
- Cosmos DB

Functions 2.0

Microsoft.NET.Sdk.Functions (.NET Standard 2.0)

- HTTP
- Timer

Microsoft.Azure.WebJobs.Extensions.Storage 3.0.0

Microsoft.Azure.WebJobs.Extensions.ServiceBus 3.0.0

Microsoft.Azure.Webjobs.Extensions.EventHubs 3.0.0

Microsoft.Azure.WebJobs.Extensions.CosmosDB 3.0.0

Microsoft.Azure.Webjobs.Extensions.EventGrid 2.0.0

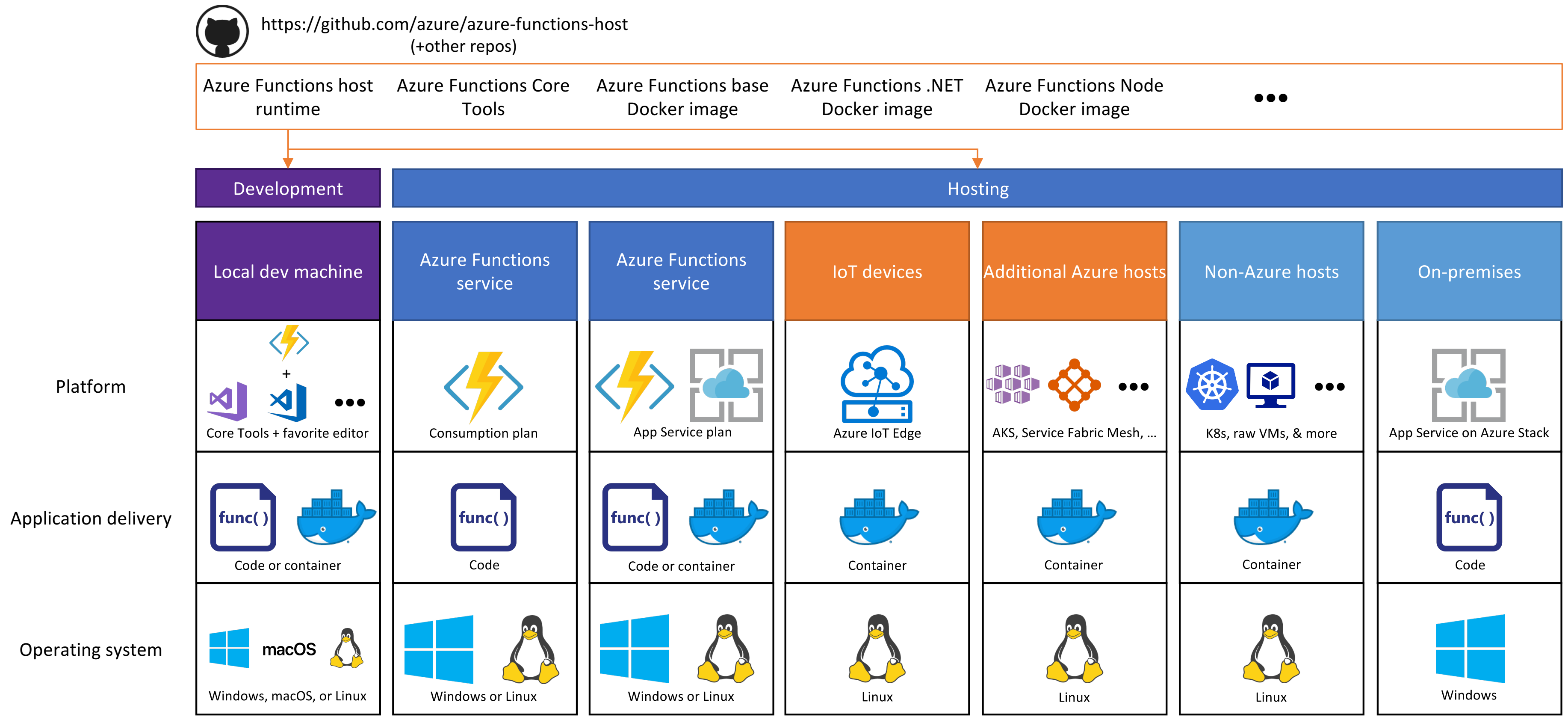
Microsoft.Azure.WebJobs.Extensions.DurableTask 1.4.0

Microsoft.Azure.Webjobs.Extensions.MicrosoftGraph 1.0.0-beta

Demo

Creating An Azure Function

Functions everywhere



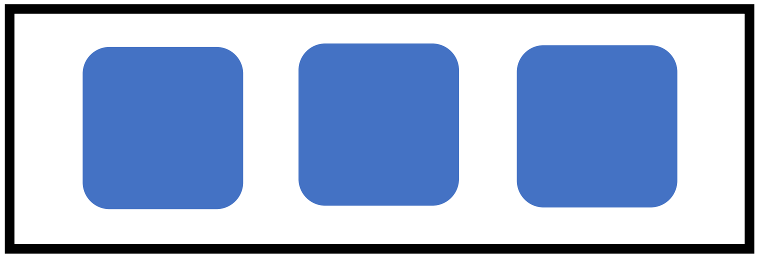
Azure Functions Hosting Options

Consumption

- Rapid scale out
- ”Unbounded” scale out
- No VNet connectivity available
- 10 minute execution
- Small instance size
- Scale to zero

App Service Plan / Environment

- Auto-scale out (~5 min)
- Fixed scale out
- VNet connectivity / hybrid
- Unlimited execution duration
- Premium instance size
- Always on



Azure Functions Hosting Options

Consumption

- Rapid scale out
- ”Unbounded” scale out
- No VNet connectivity available
- 10 minute execution
- Small instance size
- Scale to zero (cold start)

PRIVATE PREVIEW

Functions premium plan

- Rapid scale out
- “Unbounded” scale out
- VNet connectivity / hybrid
- Unlimited execution duration
- Premium instance size
- Always on

Demo

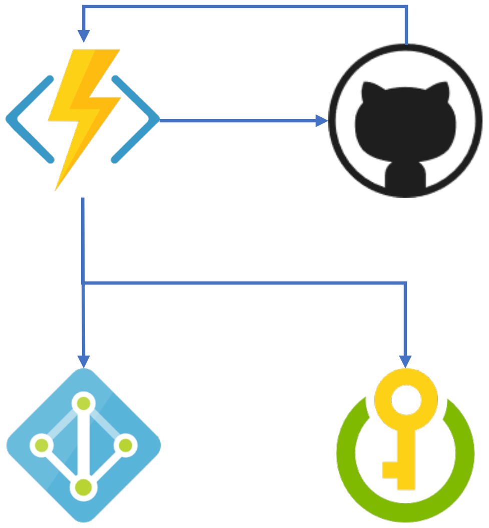
Premium Functions

The shared responsibility model

	Physical security	Host infrastructure	Operating system	Network controls	Identity & access management	Logical application server	Application logic	Client endpoints	Data classification
On-prem	⬜	⬜	⬜	⬜	⬜	⬜	⬜	⬜	⬜
IaaS	🟦	🟦	⬜	⬜	⬜	⬜	⬜	⬜	⬜
PaaS	🟦	🟦	🟦	⬕	⬕	⬕	⬜	⬜	⬜
Serverless	🟦	🟦	🟦	⬕	⬕	🟦	⬜	⬜	⬜
SaaS	🟦	🟦	🟦	🟦	🟦	🟦	🟦	⬕	⬜

🟦 Cloud provider ⬜ Cloud customer

Secrets management



```
const msRestAzure = require('ms-rest-azure');
const KeyVault = require('azure-keyvault');
const vaultUri = process.env['GITHUB_SECRET_URI'];
// Value looks like: 'https://foo.vault.azure.net/secrets/gh’

//... Getting the event

let kvToken = msRestAzure.loginWithAppServiceMSI({
    resource: 'https://vault.azure.net'
});

let keyVaultClient = new KeyVault.KeyVaultClient(kvToken);
keyVaultClient.getSecret(vaultUri).then(function (secret){
    var githubHeader = 'Basic ' + secret;
    //... Call GitHub
});
```

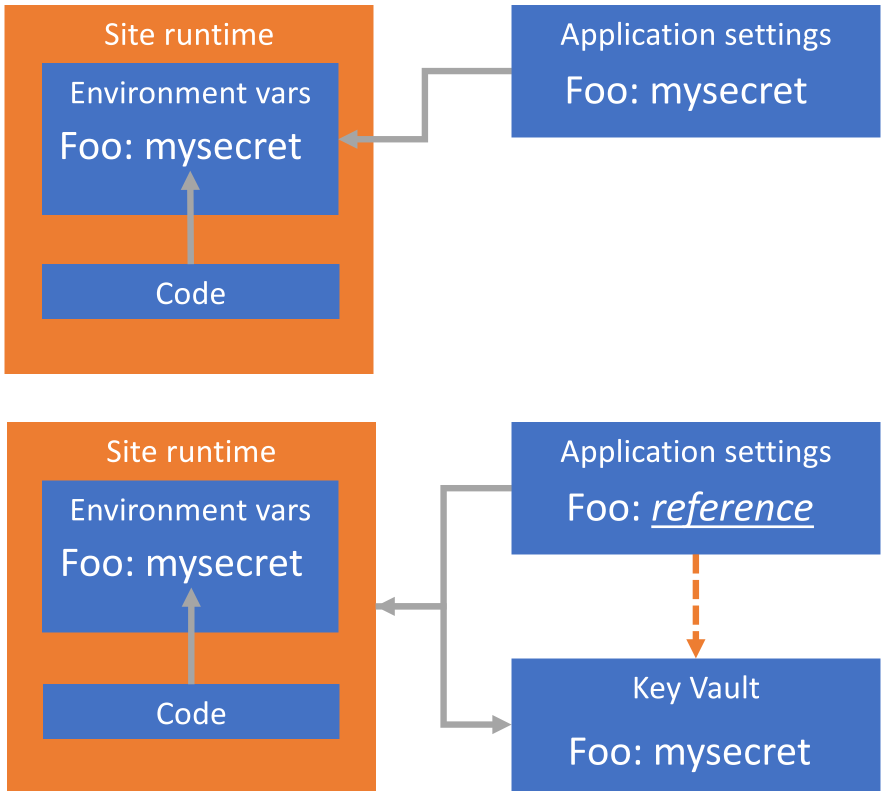
Coming soon: Key Vault references

@Microsoft.KeyVault(SecretUri=https://**myvault**.vault.azure.net/secrets/**mysecret**/**mysecretversion**)

Gets secrets out of App Settings and into secrets management

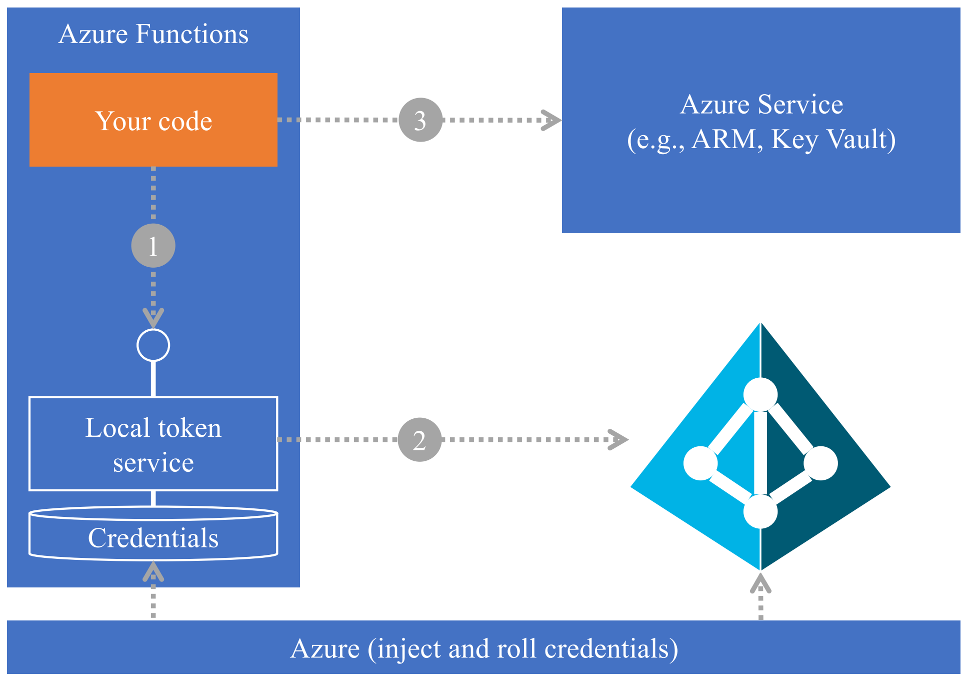
Leverages the managed identity of your function app

Versions will be required at initial preview (goal of auto-rotation)

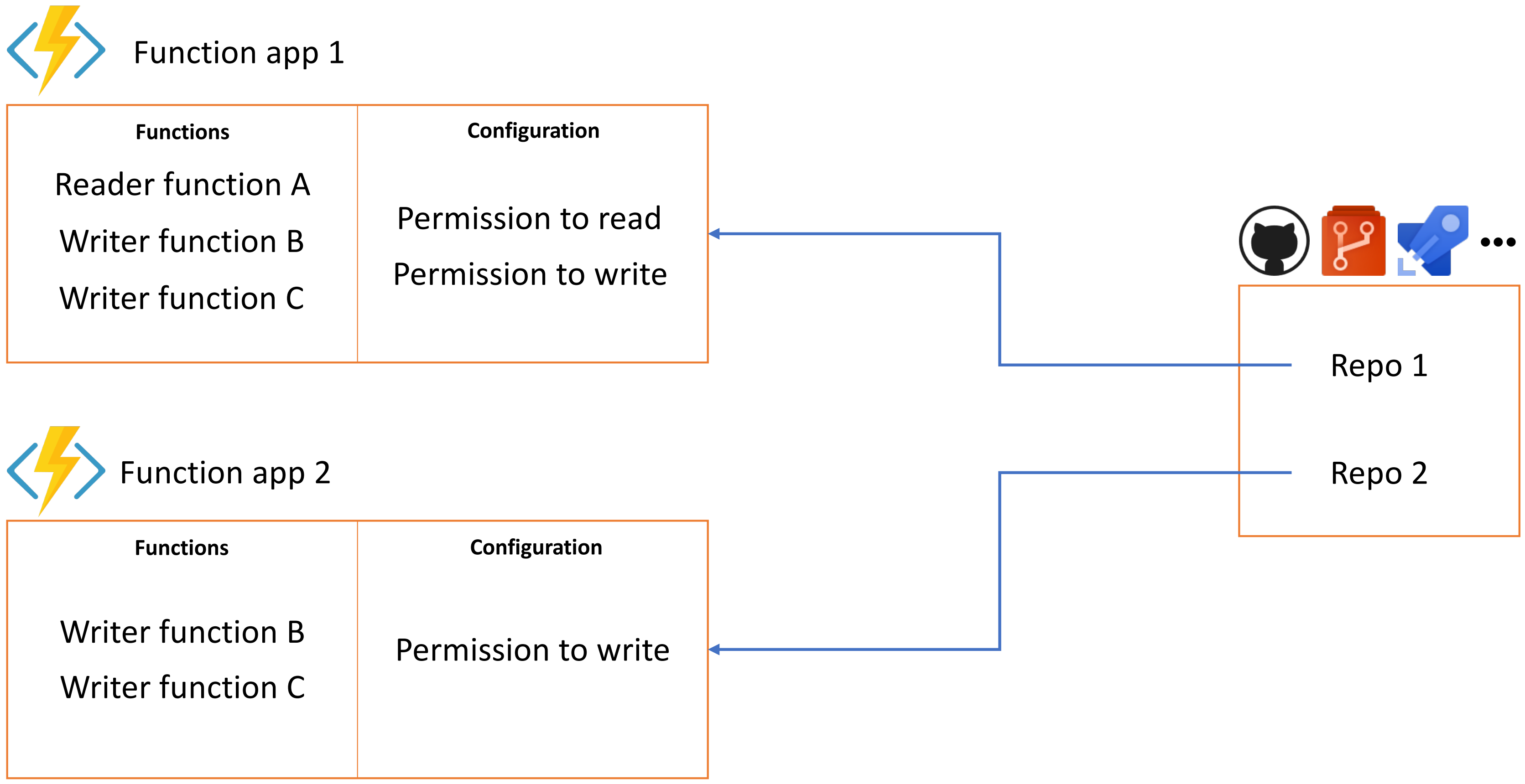


Managed identities for Azure Functions

- Keep credentials out of code
- Auto-managed identity in Azure AD for Azure resource
- Use local token endpoint to get access tokens from Azure AD
- Direct authentication with services, or retrieve creds from Azure Key Vault


Azure Functions
Your code
1
Local token service
Credentials
3
2
Azure Service (e.g., ARM, Key Vault)
Azure (inject and roll credentials)

Grouping and permissions



Spot the vulnerability!

```
module.exports = function (context, req) {
    if (req.body && req.body.name)) {
        context.res = {
            status: 202
        };
        context.bindings.outQueueMessage = {
            action: "delete",
            target: req.body.name
        };
    }
    else {
        context.res = {
            status: 400,
            body: "Please pass a name in the request body"
        };
    }
    context.done();
};
```

Meanwhile, downstream…

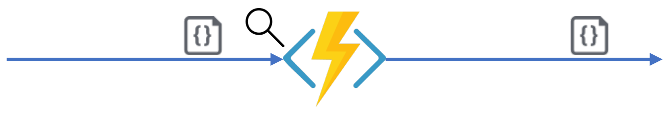
```
var Connection = require('tedious').Connection;
    var config = {
        //... Get from env vars
    };
var connection = new Connection(config);
connection.on('connect', function(err) {
    console.log("Connected");
});

//...

module.exports = function (context, myQueueItem) {
    if (myQueueItem.action === "delete") {
        let request = new Request("DELETE FROM Inventory WHERE ItemName='" + myQueueItem.target + "';", function(err) {
         if (err) {
            console.log(err);}
        });
        connection.execSql(request);
    }
    context.done();
};
```

Inputs AND outputs

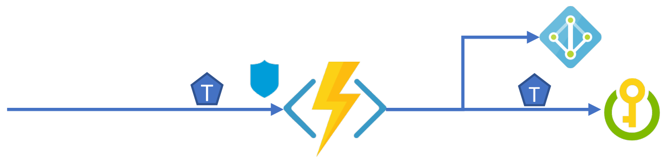
Am I validating inputs and preventing injection attacks?



Sanitization

Am I validating outputs?

Am I applying proper authorization checks?



Permissions

Am I granting proper roles and permissions? Am I enforcing least privilege?

Can my app scale well in response to new events?

Scalability

Can my downstream resources keep up with my scale?

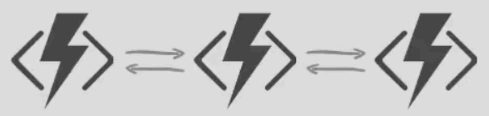
Serverless security best practices

- Standard PaaS / web app security is still a must-have
- New security tooling options needed

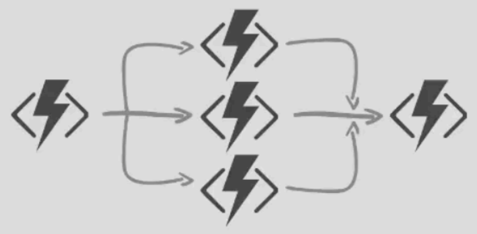
- More secrets, more secret management
- Permissions and grouping – remember least privilege
- Mind both inputs and outputs – the app is only as secure as its weakest link

- Networking solutions need development, but…

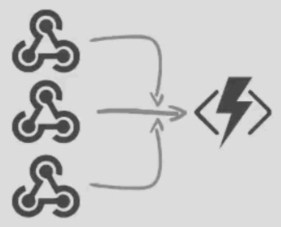
What’s still hard?



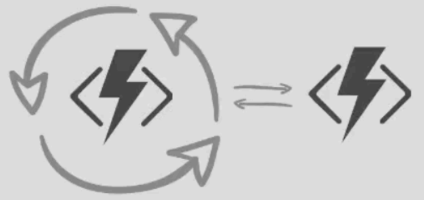
Manageable Sequencing
+ Error Handling / Compensation



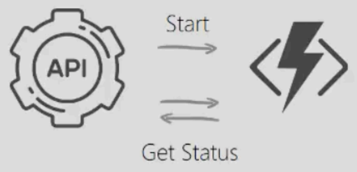
Fanning-out & Fanning-in



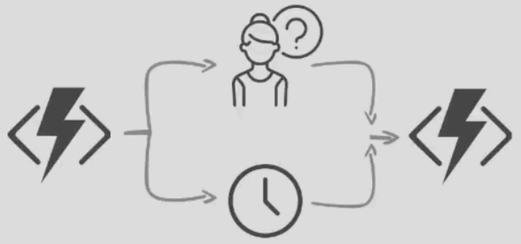
External Events Correlation



Flexible Automated Long-running
Process Monitoring



Http-based
Async Long-running APIs

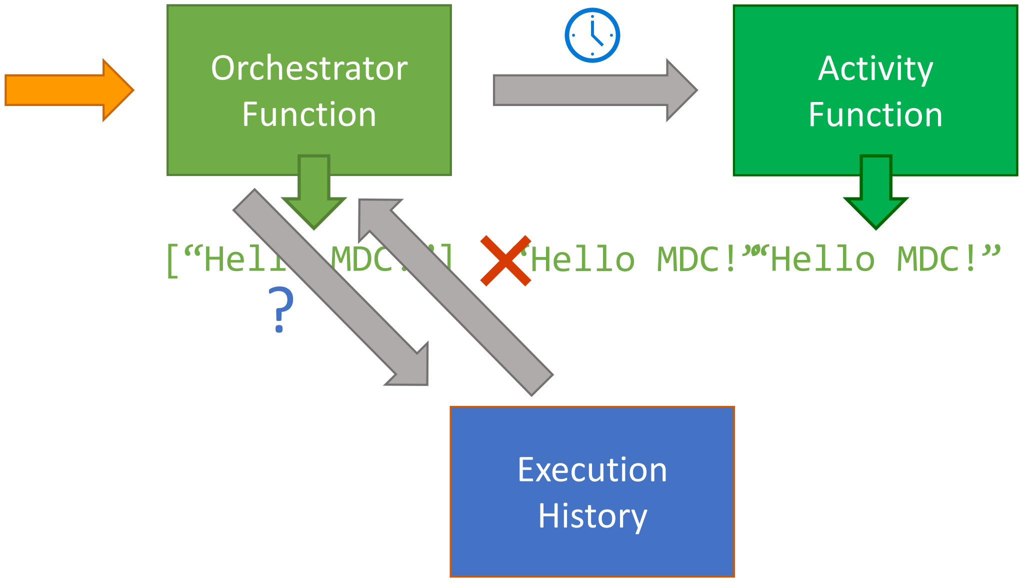


Human Interaction

```
var outputs = new List<string>();

outputs.Add(await context.CallActivityAsync<string>(“Hello”, “MDC”));

return outputs;
```



History Table

Orchestrator Started
Execution Started
Task Scheduled, Hello, “MDC”
Orchestrator Completed
Task Completed, “Hello MDC!”
Orchestrator Started
Execution Completed, ["Hello MDC!"]
Orchestrator Completed

Demo

Durable Functions

Questions

Serverless security best practices

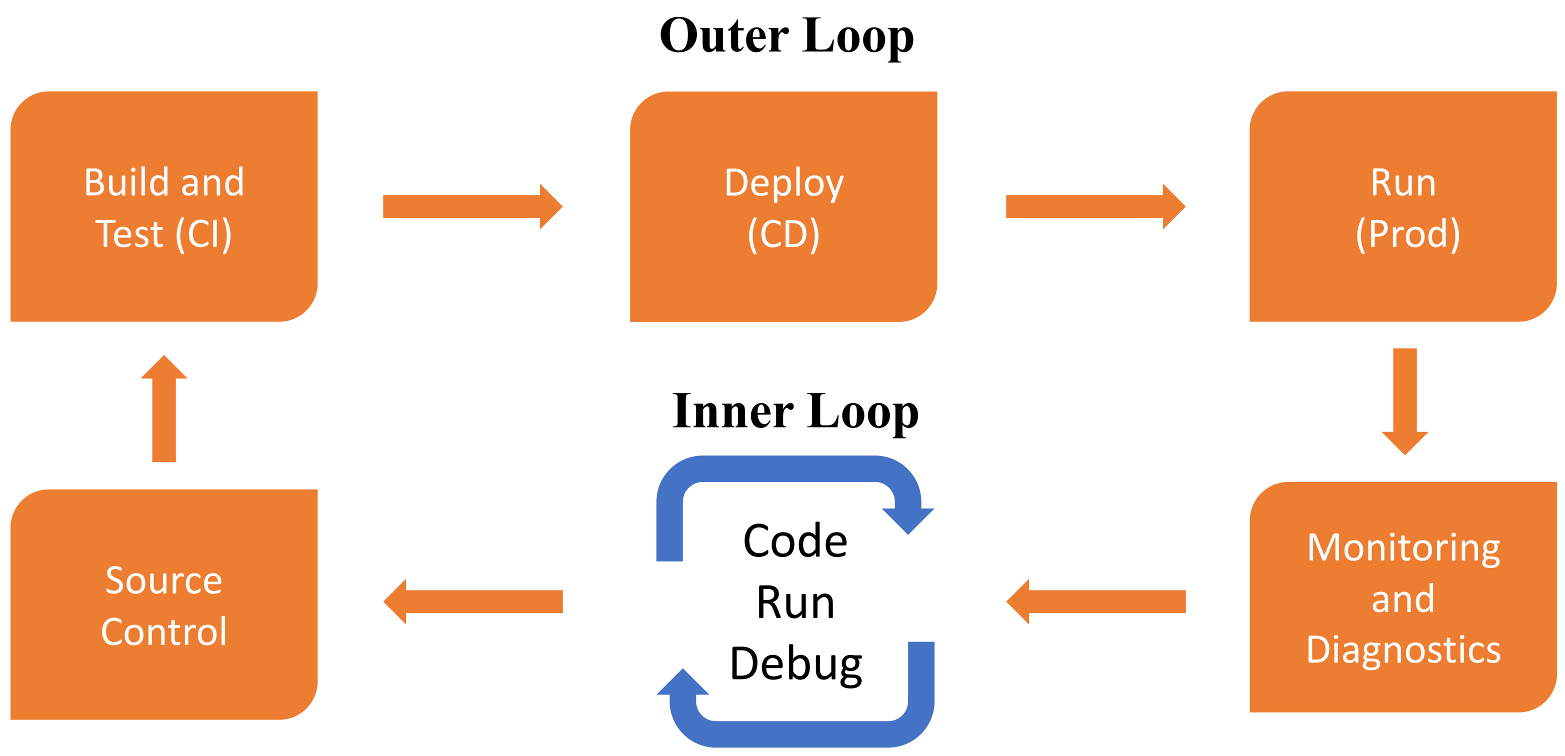
- Standard PaaS / web app security is still a must-have
- New security tooling options needed

- More secrets, more secret management
- Permissions and grouping – remember least privilege
- Mind both inputs and outputs – the app is only as secure as its weakest link

- Networking solutions need development, but…

Thank you!

Inner and Outer Loop Development



Available tools of Azure Functions

Local Tools	**Deployment Center (Kudu)**	**Azure DevOps**	**Other CI/CD**
Quickly publish to production	App Services powered CI/CD	Fully managed CI/CD	Any other CI/CD tool (Jenkins, Octopus, Travis)
Best Suited – Quickly validate code works in the cloud	Best Suited – One-click deploy from GitHub/source	Best Suited – Production CI/CD with various environments	Best Suited – Integrated serverless with existing tools and processes
Watch out – “Friends don’t let friend right-click publish”	Watch out – Not as customizable as Azure DevOps pipelines	Watch out – Web Deploy vs Run from Package	Watch out – Documentation and samples are limited
Tip – Use the ‘run from package’ feature	Tip – Use the new “Deployment Center” section	Tip – Can call functions as release gates	Tip – Use the ‘run from package’ publish gesture