TOPIC

Azure Functions best practices

Tips when going to production

Who I am

Microsoft® MVP Most Valuable Professional

Stefano Demiliani

CTO, Azure Solution Architect

@demiliani

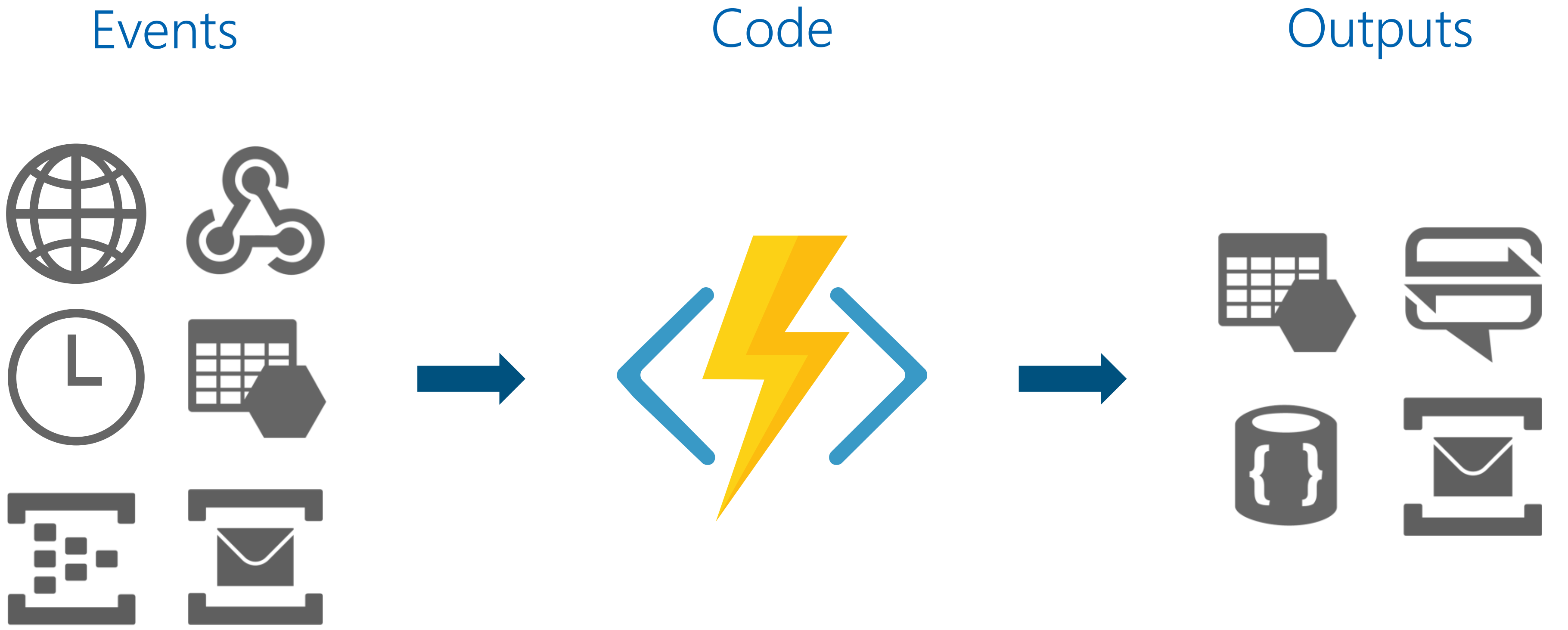
https://github.com/demiliani

https://www.linkedin.com/in/stefano-demiliani/

Azure Functions

Fundamental building block for creating cloud apps!

Process events with Serverless code.



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

Easy monitoring

Send results to an ever-growing collection of services

Available tiers

- App Service offers dedicated and dynamic tiers.
- Dedicated:
  - Basic, Standard, Premium
  - Pay based on # of reserved VMs
  - You’re responsible for scale
- Dynamic:
  - Pay on number of executions
  - Platform responsible for scale (instances of the Azure Functions host are dynamically added and removed based on the number of incoming events)

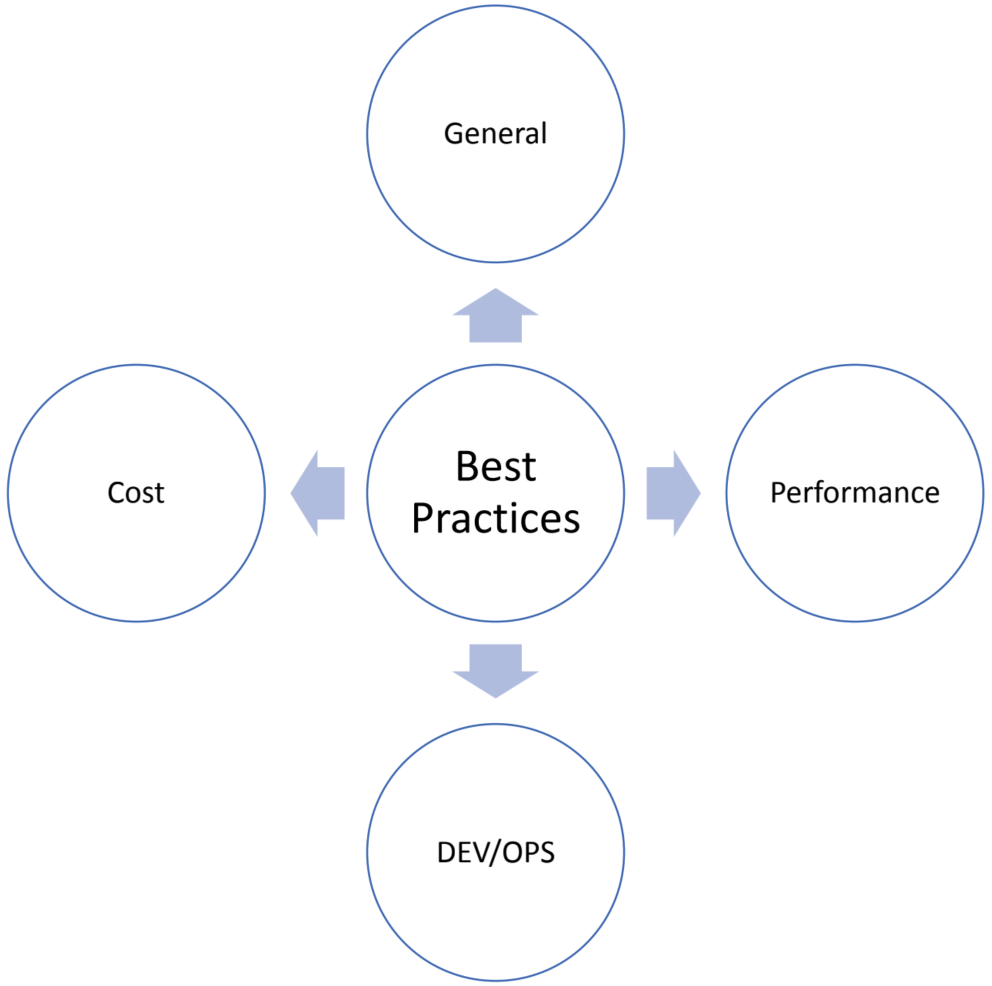
Dynamic tier pricing

Pay per execution model - two meters, three units

- Number of executions
- Duration of execution x reserved memory

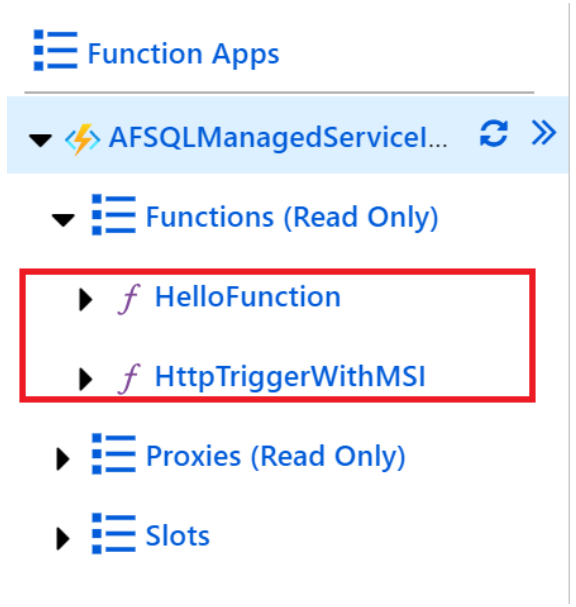
- **Executions**
  - total number of requested executions each month for all functions
  - The first million executions are included **free each month**.
- **Resource consumption**
  - “**Observed resource consumption**” measured in gigabyte seconds (GB-s).
    - Observed resource consumption is calculated by multiplying average memory size in gigabytes by the time in milliseconds it takes to execute the function.
    - Memory used by a function is measured by rounding up to the nearest 128 MB, up to the maximum memory size of 1,536 MB
    - Execution time calculated by rounding up to the nearest 1 ms.
    - The minimum execution time and memory for a single function execution is 100 ms and 128 mb respectively.
    - Functions pricing includes a **monthly free** grant of 400,000 GB-s.

Best practices when going to production



TIP 1: Multiple Function apps with few Functions

- Azure *Function App* is a container for one or more *functions* – each interacting with other Azure Services, APIs or applications



- The function can be part of an overall serverless architecture, where components can still fail.

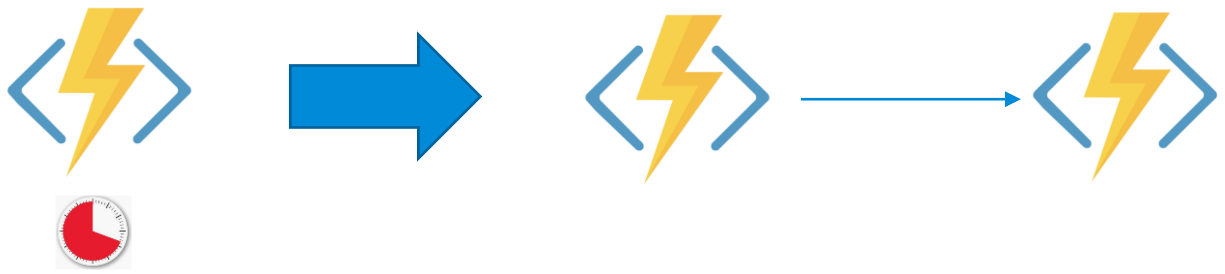
To minimize the impact to your functions, it's recommended to <u>split them up into multiple Function Apps to make them independently deployable and thus independently failable</u>.

TIP 2: Avoid Long Running Functions

- Large, long-running functions can cause unexpected timeout issues

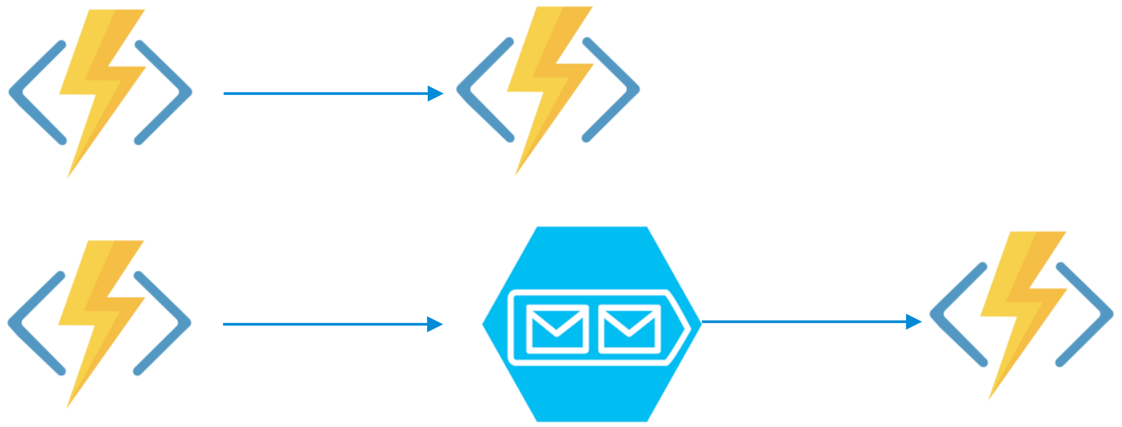
Plan	Runtime Version	Default	Maximum
Consumption	1.x	5	10
Consumption	2.x	5	10
Consumption	3.x	5	10
Premium	1.x	30	Unlimited
Premium	2.x	30	Unlimited
Premium	3.x	30	Unlimited
App Service	1.x	Unlimited	Unlimited
App Service	2.x	30	Unlimited
App Service	3.x	30	Unlimited

- Importing dependencies can also cause increased load times that result in unexpected timeouts. Dependencies are loaded both explicitly and implicitly.
- Whenever possible, refactor large functions into smaller function sets that work together and return responses fast.



TIP 3: Handle cross function communication efficiently

- Function to Function communications on long running flows should be avoided

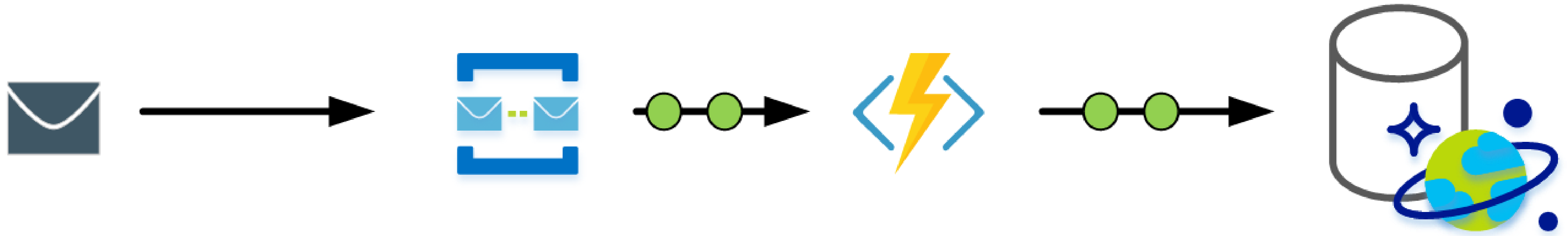


- Cross-function communication:
  - Storage queues: cheaper, individual message limited to 64 Kb
  - Azure Service Bus: messages up to 256 KB in Standard tier, up to 1 MB in Premium tier, Topics useful for message filtering
  - Event Hub for high volume communications
- Durable Functions and Azure Logic Apps are built to manage state transitions and communication between multiple functions.

TIP 4: Write functions to be stateless

State informations shoud be associated to your data and a function should be stateless. Do not store state informations within the application itself.

For example, an order being processed would likely have an associated `state` member. A function could process an order based on that state while the function itself remains stateless.



In a serverless architecture, your functions should "do one thing", be stateless and idempotent, and finish as quickly as possible.

TIP 5: Write defensive functions and validate inputs

Handle exceptions and design your functions with the ability to continue from a previous failure point during the next execution.

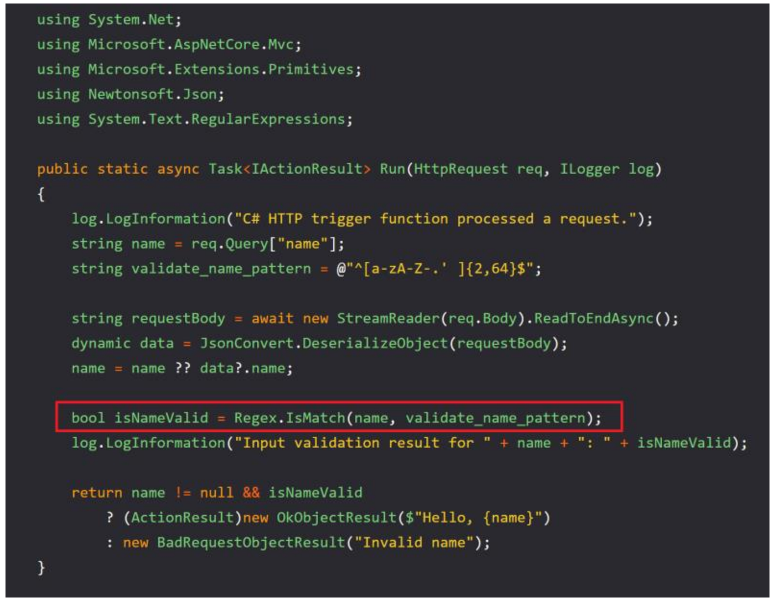
Example:

Step 1: query DB and retrieving 10000 rows

Step 2: create a message in a Queue for each row to process

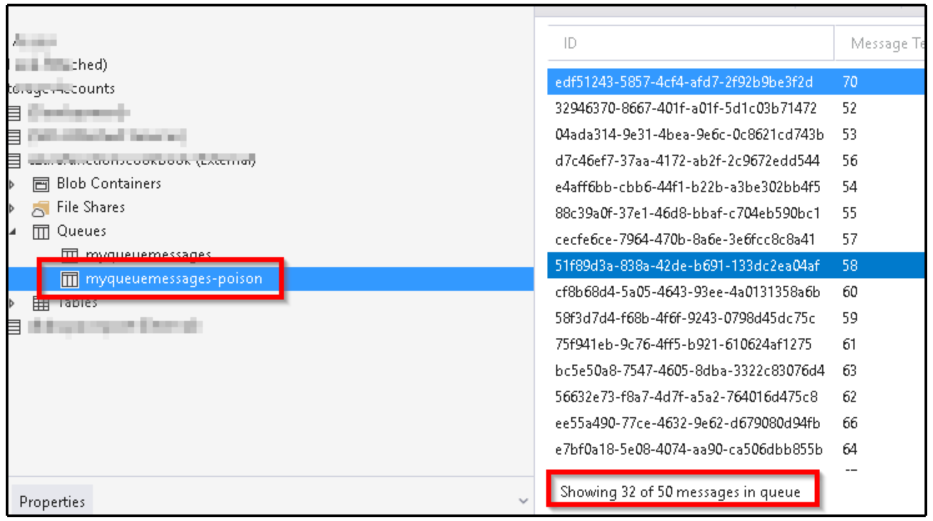
Step 3: Exception occours on inserting row 5000 → Track items in a set that you've completed. Otherwise, you might insert them again next time.

Always validate your functions input to avoid injection flaws



```
using System.Net;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Extensions.Primitives;
using Newtonsoft.Json;
using System.Text.RegularExpressions;

public static async Task<IActionResult> Run(HttpRequest req, ILogger log)
{
    log.LogInformation("C# HTTP trigger function processed a request.");
    string name = req.Query["name"];
    string validate_name_pattern = @"^[a-zA-Z-.' ]{2,64}$";

    string requestBody = await new StreamReader(req.Body).ReadToEndAsync();
    dynamic data = JsonConvert.DeserializeObject(requestBody);
    name = name ?? data?.name;

    bool isNameValid = Regex.IsMatch(name, validate_name_pattern);
    log.LogInformation("Input validation result for " + name + ": " + isNameValid);

    return name != null && isNameValid
        ? (ActionResult)new OkObjectResult($"Hello, {name}")
        : new BadRequestObjectResult("Invalid name");
}
```

If a function triggered by a *QueueTrigger* fails, the Azure Functions runtime automatically retries that function five times for that specific queue message. If the message continues to fail, then the bad message is moved to a "**poison**" **queue**.



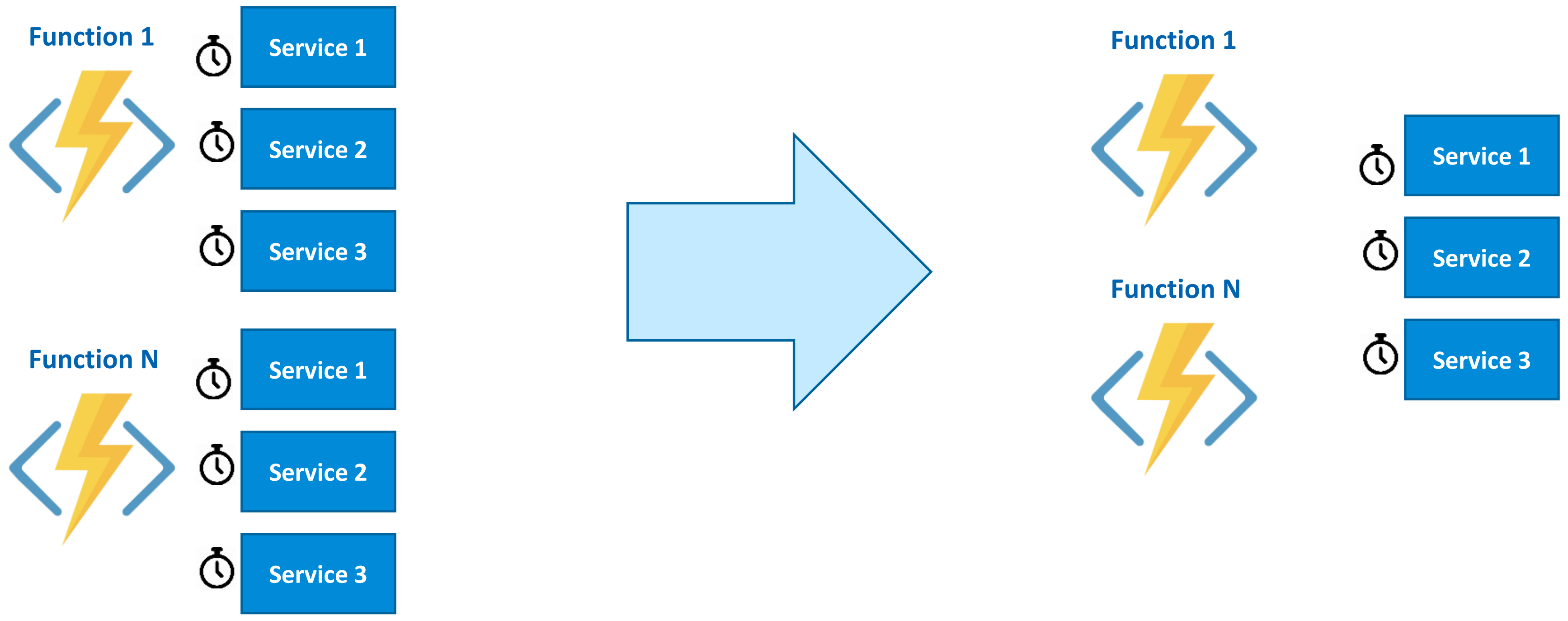
ID	Message Te
edf51243-5857-4cf4-afd7-2f92b9be3f2d	70
32946370-8667-401f-a01f-5d1c03b71472	52
04ada314-9e31-4bea-9e6c-0c8621cd743b	53
d7c46ef7-37aa-4172-ab2f-2c9672edd544	56
e4aff6bb-cbb6-44f1-b22b-a3be302bb4f5	54
88c39a0f-37e1-46d8-bbaf-c704eb590bc1	55
cecfe6ce-7964-470b-8a6e-3e6fcc8c8a41	57
51f89d3a-838a-42de-b691-133dc2ea04af	58
cf8b68d4-5a05-4643-93ee-4a0131358a6b	60
58f3d7d4-f68b-4f6f-9243-0798d45dc75c	59
75f941eb-9c76-4ff5-b921-610624af1275	61
bc5e50a8-7547-4605-8dba-3322c83076d4	63
56632e73-f8a7-4d7f-a5a2-764016d475c8	62
ee55a490-77ce-4632-9e62-d679080d94fb	66
e7bf0a18-5e08-4074-aa90-ca506dbb855b	64

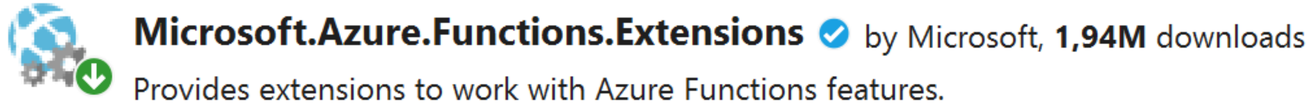
Showing 32 of 50 messages in queue

TIP 6: Dependency Injection

Scenario: Function app with N functions deployed that need to use same services.

Don't create N instances of these services but reuse the same instance (instantiate a service once when the Function App (host) starts up, and re-use that object across all executions of your Functions inside this Function App).




Microsoft.Azure.Functions.Extensions by Microsoft, **1,94M** downloads
Provides extensions to work with Azure Functions features.

DEMO

TIP 7: Choose the right plan for your scalability needs

When you create a function app in Azure, you must choose a hosting plan for your app. There are three hosting plans available for Azure Functions: Consumption plan, Premium plan, and Dedicated (App Service) plan.

The *Consumption* plan is the default hosting plan and offers the following benefits:

- Pay only when your functions are running
- Scale out automatically, even during periods of high load

Function apps in the same region can be assigned to the same Consumption plan:

- No downside or impact to having multiple apps running in the same Consumption plan.
- Assigning multiple apps to the same Consumption plan has no impact on resilience, scalability, or reliability of each app.

Billing: per execution and memory consumed

Cold start

Scalability tips

Premium plan supports the following features:

- Perpetually warm instances to avoid any cold start
- Virtual Network connectivity
- Unlimited execution duration (60 minutes guaranteed)
- Premium instance sizes (one core, two core, and four core instances)
- More predictable pricing
- High-density app allocation for plans with multiple function apps

Billing: based on the number of core seconds and memory used across needed and pre-warmed instances. At least one instance must be warm at all times per plan. This means that there is a minimum monthly cost per active plan, regardless of the number of executions.

Consider the Azure Functions Premium plan in the following situations:

- Your function apps run continuously, or nearly continuously.
- You have a high number of small executions and have a high execution bill but low GB second bill in the Consumption plan.
- You need more CPU or memory options than what is provided by the Consumption plan.
- Your code needs to run longer than the *maximum execution time allowed* on the Consumption plan.
- You require features that are only available on a Premium plan, such as virtual network connectivity.

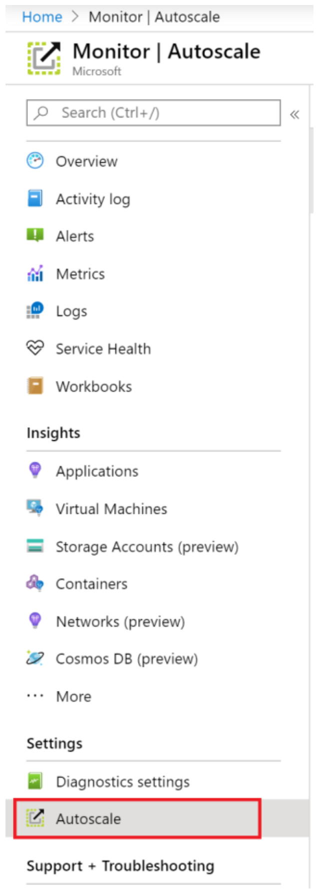
Scalability tips

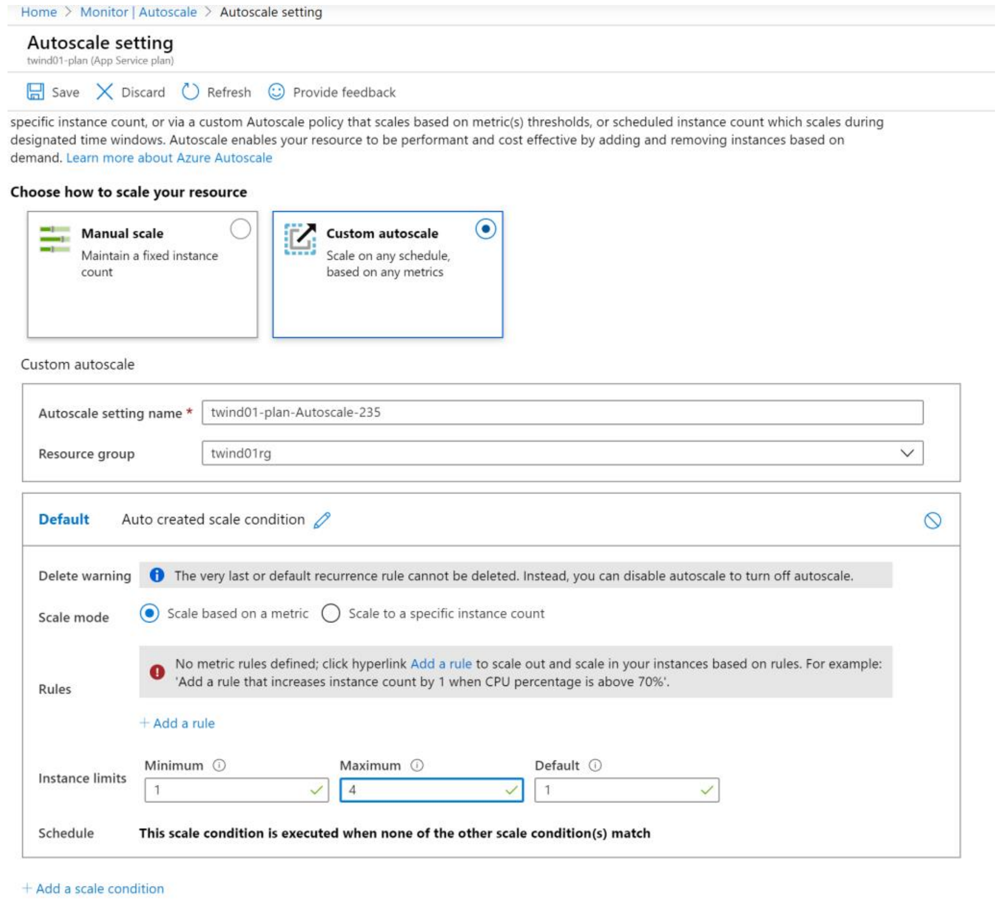
Consider an *App Service* plan in the following situations:

- You have existing, underutilized VMs that are already running other App Service instances.
- You want to provide a custom image on which to run your functions.

Scaling:

- manually scale out by adding more VM instances
- scale up by choosing a different App Service plan
- Enable autoscale





Autoscale setting
twind01-plan (App Service plan)

specific instance count, or via a custom Autoscale policy that scales based on metric(s) thresholds, or scheduled instance count which scales during designated time windows. Autoscale enables your resource to be performant and cost effective by adding and removing instances based on demand. Learn more about Azure Autoscale

Choose how to scale your resource

Manual scale Maintain a fixed instance count

[✓ **Custom autoscale** Scale on any schedule, based on any metrics]

Custom autoscale

Autoscale setting name *	twind01-plan-Autoscale-235
Resource group	twind01rg

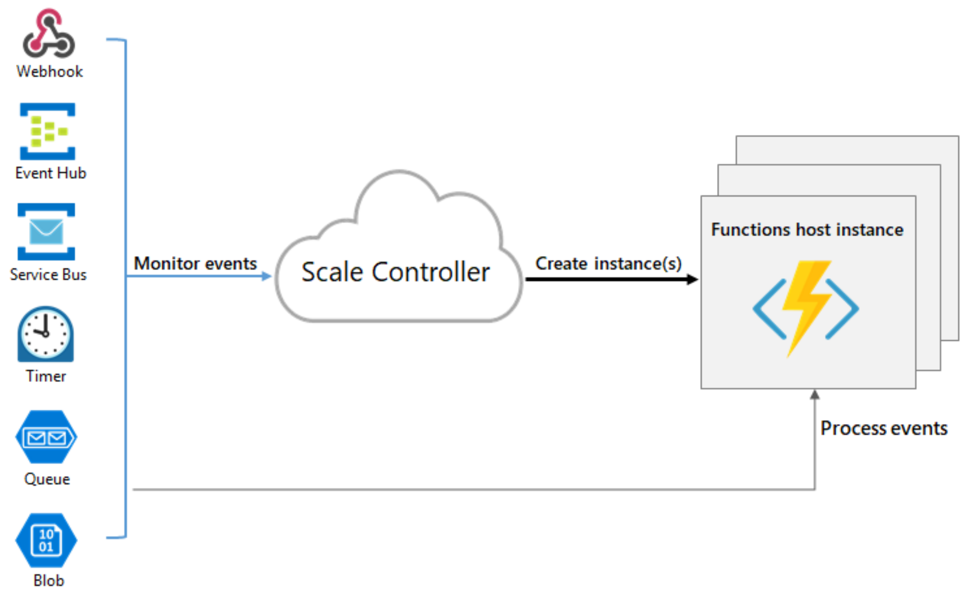
Default Auto created scale condition

Delete warning	The very last or default recurrence rule cannot be deleted. Instead, you can disable autoscale to turn off autoscale.
Scale mode	[✓ Scale based on a metric] Scale to a specific instance count
Rules	No metric rules defined; click hyperlink Add a rule to scale out and scale in your instances based on rules. For example: 'Add a rule that increases instance count by 1 when CPU percentage is above 70%'. + Add a rule
Instance limits	Minimum 1 · Maximum 4 · Default 1
Schedule	**This scale condition is executed when none of the other scale condition(s) match**

+ Add a scale condition

TIP 8: code tips for the scale controller

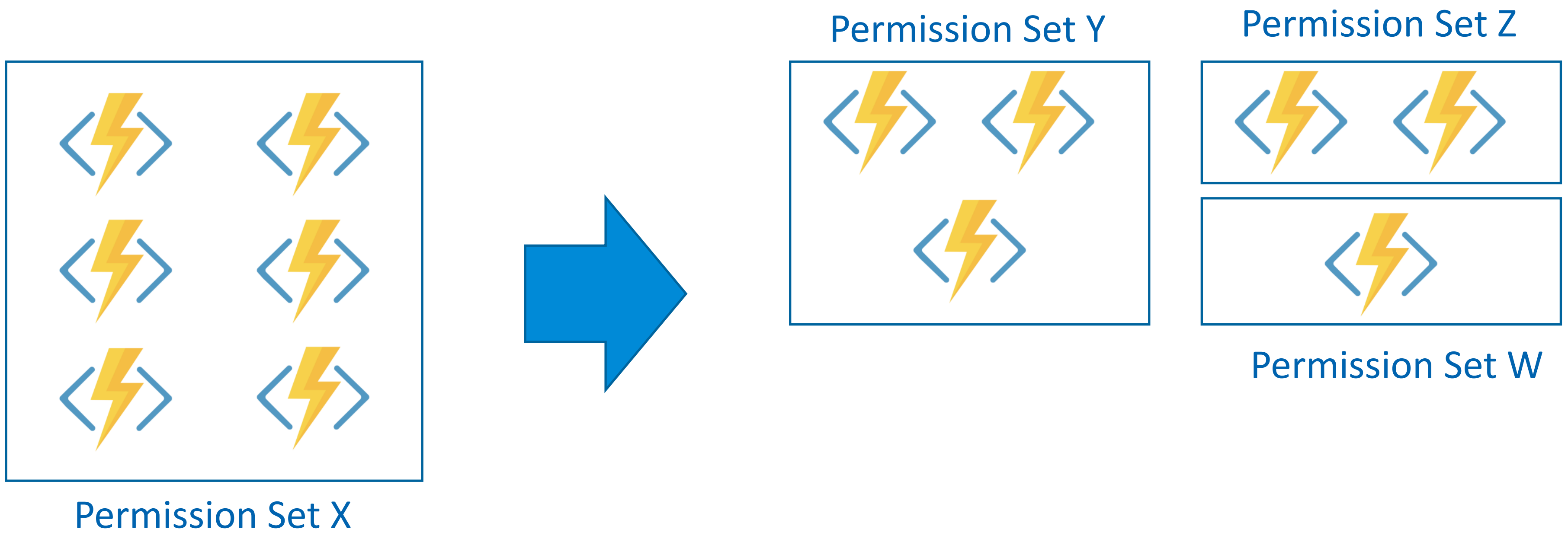
Azure Functions uses a component called the *scale controller* to monitor the rate of events and determine whether to scale out or scale in. The unit of scale for Azure Functions is the function app.



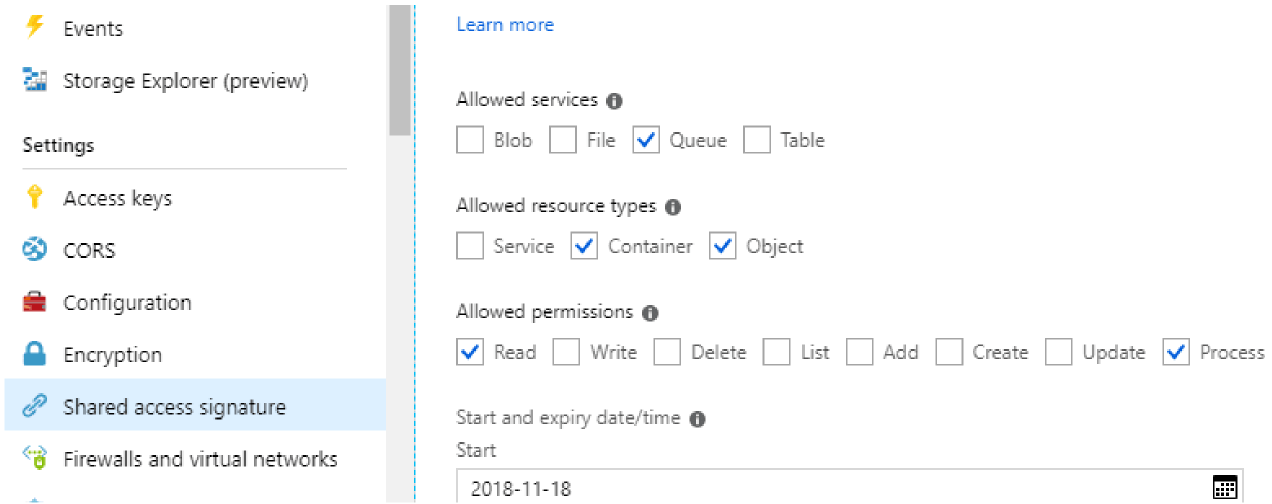
- Reuse connections to external resources whenever possible
- Don’t share storage accounts between function apps
- Use async programming
- Receive messages in batch when possible
- Use multiple worker processes in app settings (maximum number of processes that our functions runtime can invoke within a single machine): **FUNCTIONS_WORKER_PROCESS_COUNT** (1 to 10)

TIP 9: least privilege

Follow *Least Privilege* principle!



Use **Shared Access Signature (SAS)** tokens to get limited access to other resources and services



Allowed services
[ ] Blob [ ] File [✓ Queue] [ ] Table

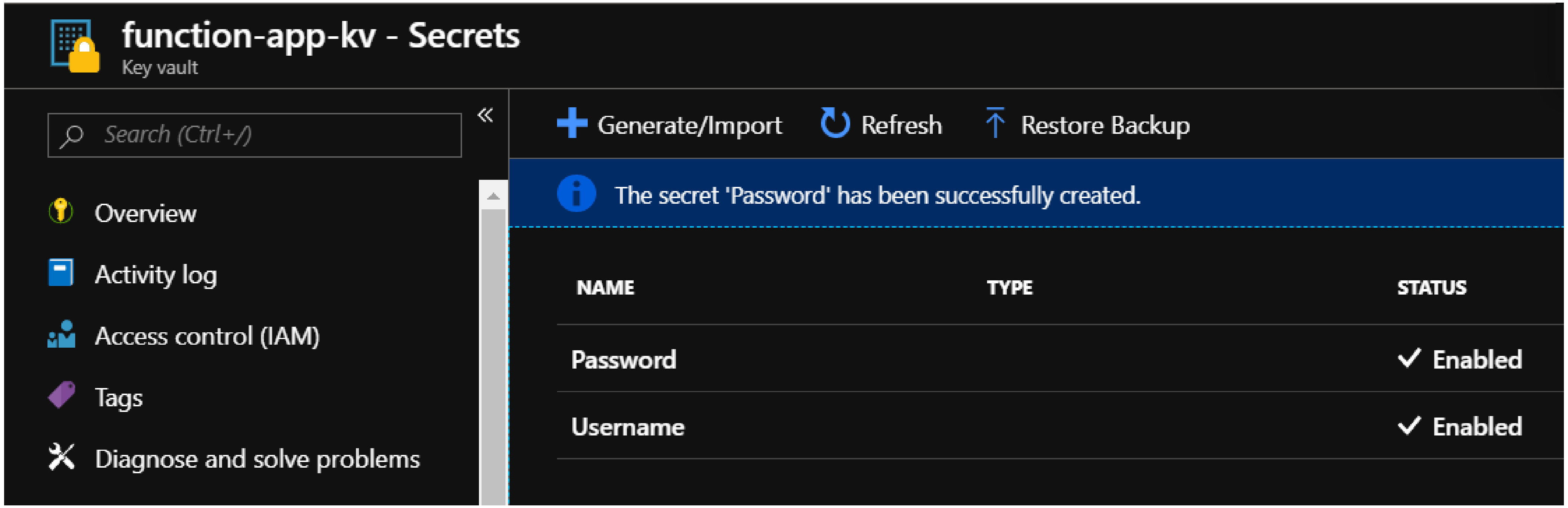
Allowed resource types
[ ] Service [✓ Container] [✓ Object]

Allowed permissions
[✓ Read] [ ] Write [ ] Delete [ ] List [ ] Add [ ] Create [ ] Update [✓ Process]

Start and expiry date/time
Start
2018-11-18

TIP 10: Use Azure Key Vault

Create Azure Key Vault instance with secrets:



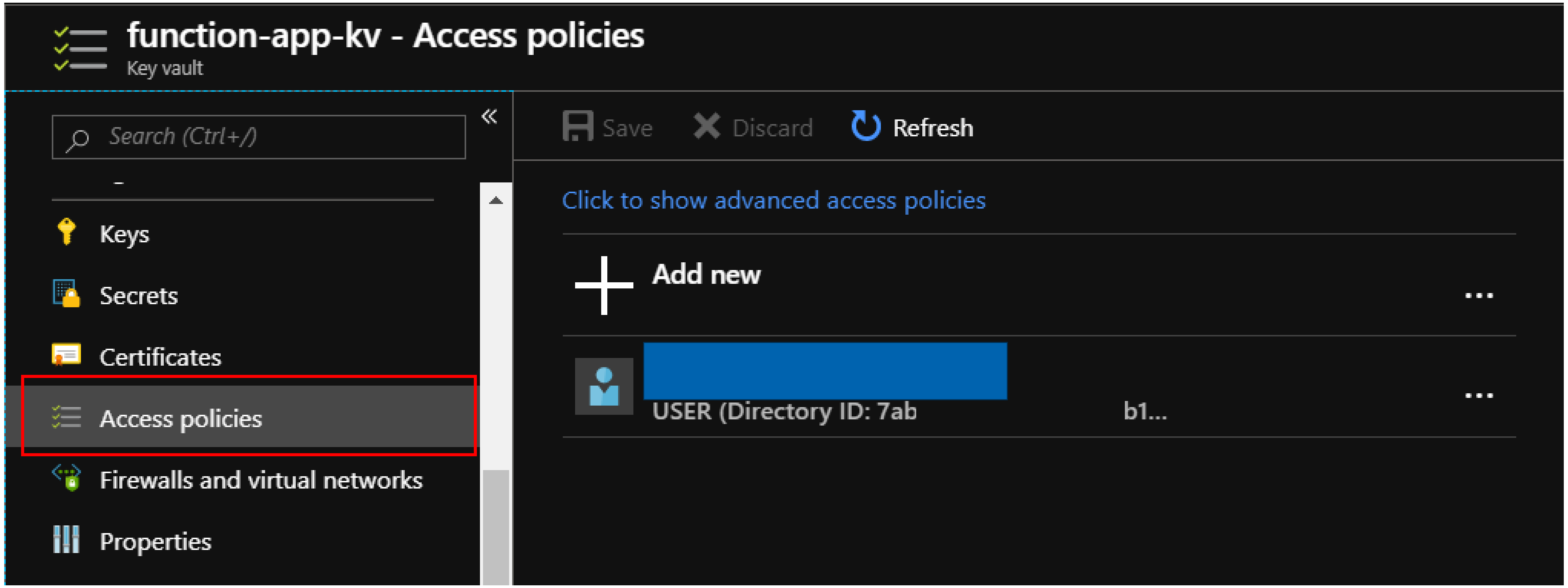
function-app-kv - Secrets
Key vault

The secret 'Password' has been successfully created.

NAME	TYPE	STATUS
Password		✓ Enabled
Username		✓ Enabled

Azure Key Vault

Add Key Vault access policy for the Function App:



function-app-kv - Access policies
Key vault

Search (Ctrl+/)

- Keys
- Secrets
- Certificates
- Access policies
- Firewalls and virtual networks
- Properties

Save | Discard | Refresh

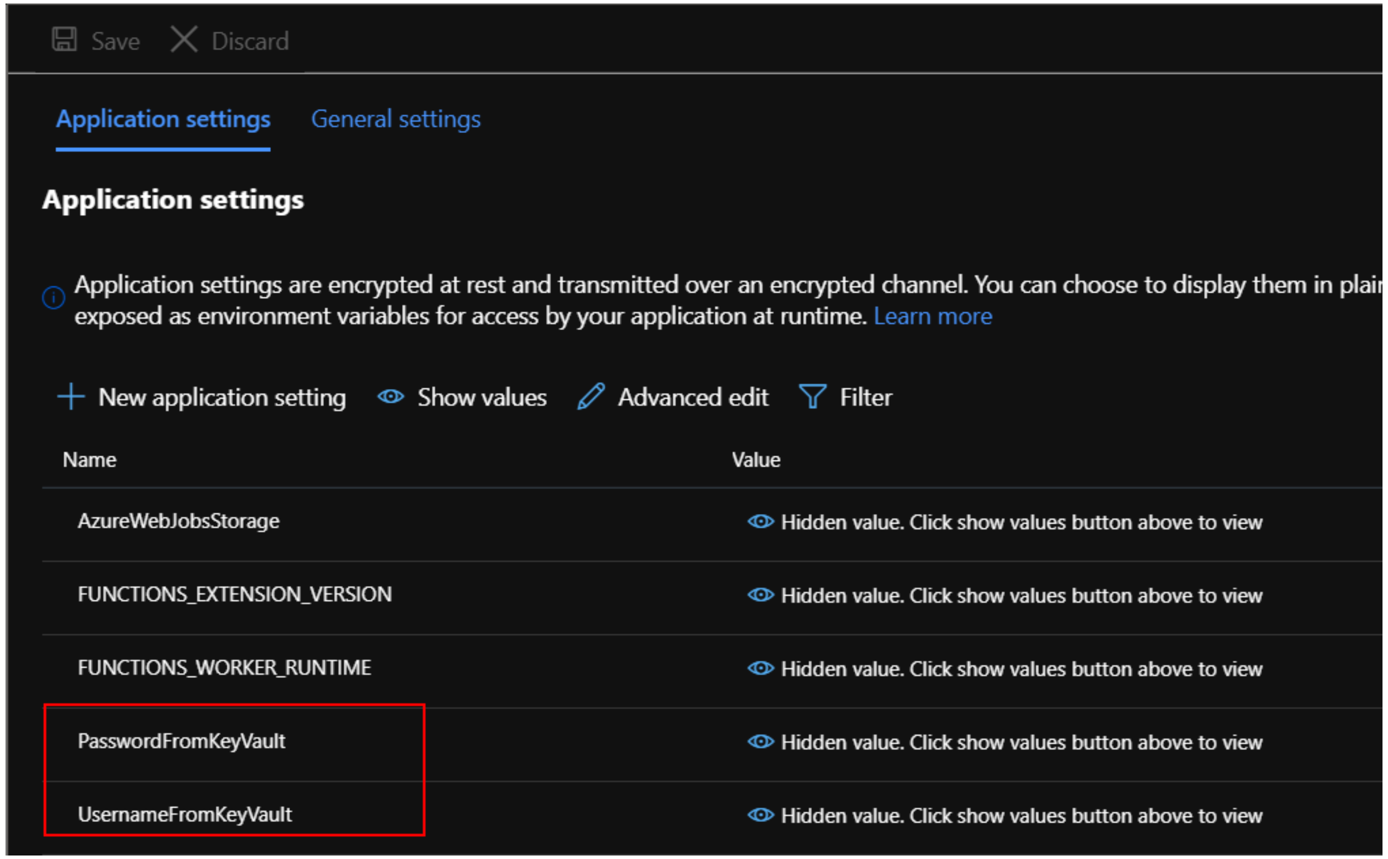
Click to show advanced access policies

Add new ...

[redacted]
USER (Directory ID: 7ab b1... ...

Azure Key Vault

Modify Function App configuration for using Key Vault secrets:



Application settings

Application settings are encrypted at rest and transmitted over an encrypted channel. You can choose to display them in plai
exposed as environment variables for access by your application at runtime. Learn more

Name	Value
AzureWebJobsStorage	Hidden value. Click show values button above to view
FUNCTIONS_EXTENSION_VERSION	Hidden value. Click show values button above to view
FUNCTIONS_WORKER_RUNTIME	Hidden value. Click show values button above to view
PasswordFromKeyVault	Hidden value. Click show values button above to view
UsernameFromKeyVault	Hidden value. Click show values button above to view

PasswordFromKeyVault: @Microsoft.KeyVault(SecretUri={copied identifier for the password secret})

Azure Key Vault

You can use Key Vault secrets in your function code:

```
var username = Environment.GetEnvironmentVariable("UsernameFromKeyVault",
                                        EnvironmentVariableTarget.Process);
var password = Environment.GetEnvironmentVariable("PasswordFromKeyVault",
                                        EnvironmentVariableTarget.Process);

log.LogInformation($"Username: {username}");
log.LogInformation($"Username: {password}");
```

Azure App Configuration is a cloud service that is used to store non-secret configuration settings in a centralised location

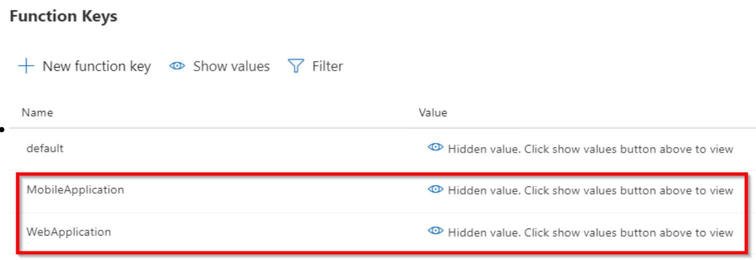
Azure Key Vault complements *Azure App Configuration* by being the configurable and secure place that we should use for application secrets.

TIP 11: Authorization and Authentication

You should use **Authorization** in development and testing scenarios only and use **Authentication** and access restrictions in any type of production workload.

Authorization scopes:

- *Function*: Requires a specific key to request a specific function (HTTP triggered).
- *Host*: Requires a specific key, but can trigger any function in the Function App.



Function Keys

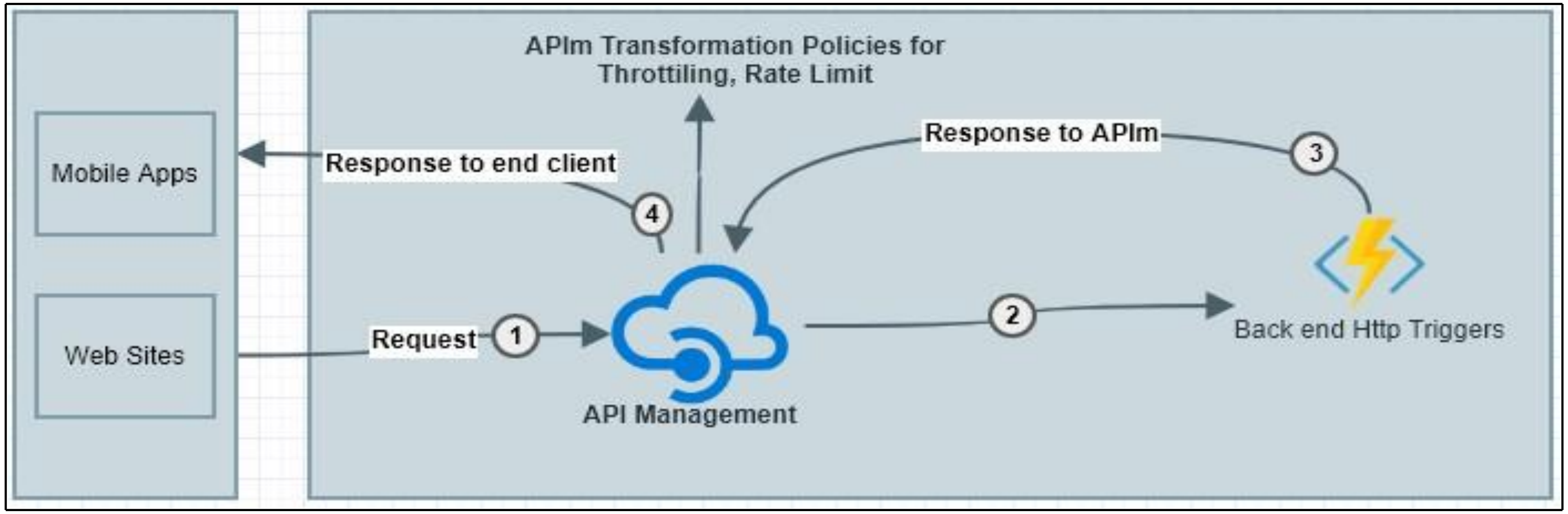
+ New function key · Show values · Filter

Name	Value
default	Hidden value. Click show values button above to view
MobileApplication	Hidden value. Click show values button above to view
WebApplication	Hidden value. Click show values button above to view

For production workloads:

- Turn on App Service auth to enable Azure AD (or any third party auth provider) to authenticate the clients calling your function.
- Use API Management.
- Use an Azure App Service Environment (ASE), which also enables you to use a WAF (Web Application Firewall).
- Use an App Service Plan where you restrict access and implement Azure Front Door + WAF to handle your incoming requests.
- Require clients to authenticate with client certificates.

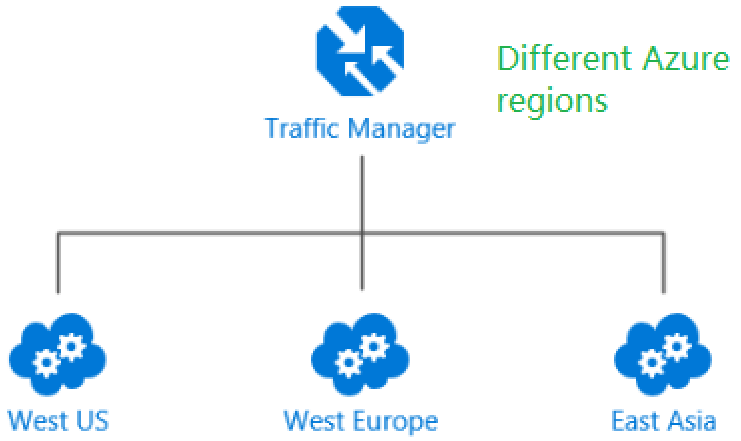
Authorization and Authentication

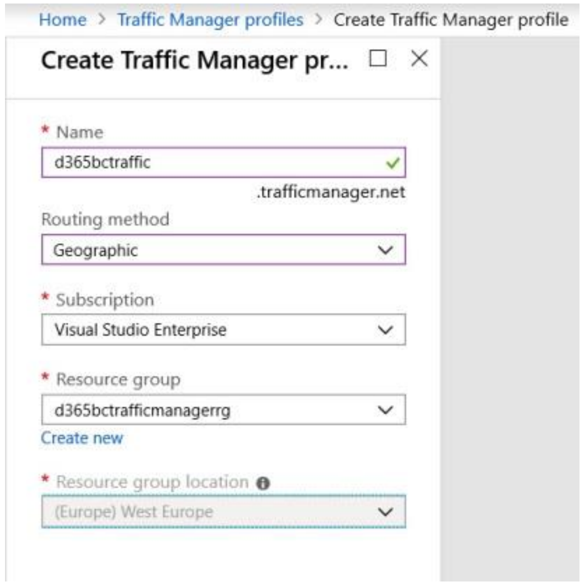


DEMO

TIP 12: Traffic Manager

Azure Traffic Manager is a DNS-based traffic load balancer that enables you to distribute traffic optimally to services across global Azure regions, while providing high availability and responsiveness.




Home > Traffic Manager profiles > Create Traffic Manager profile

Create Traffic Manager pr...

* Name
d365bctraffic
.trafficmanager.net

Routing method
Geographic

* Subscription
Visual Studio Enterprise

* Resource group
d365bctrafficmanagerrg
Create new

* Resource group location
(Europe) West Europe

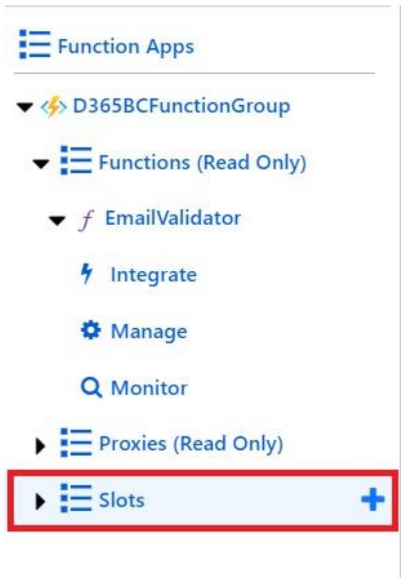
Traffic Routing methods between endpoints: weighted, priority, performance, geographic, based on IP addresses.

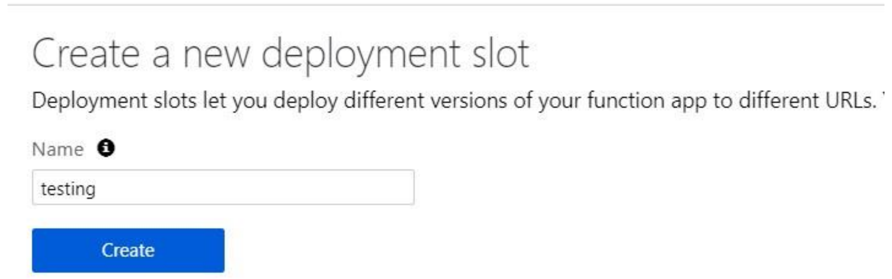
Azure Traffic Manager includes **built-in endpoint monitoring and automatic endpoint failover**. This feature helps you deliver high-availability applications that are resilient to endpoint failure, including Azure region failures.

http://d365bctraffic.trafficmanager.net/api/emailvalidation?name=stefano

TIP 13: Deployment Slots

Use deployment slots for testing your "preview" functions with customers without affecting others

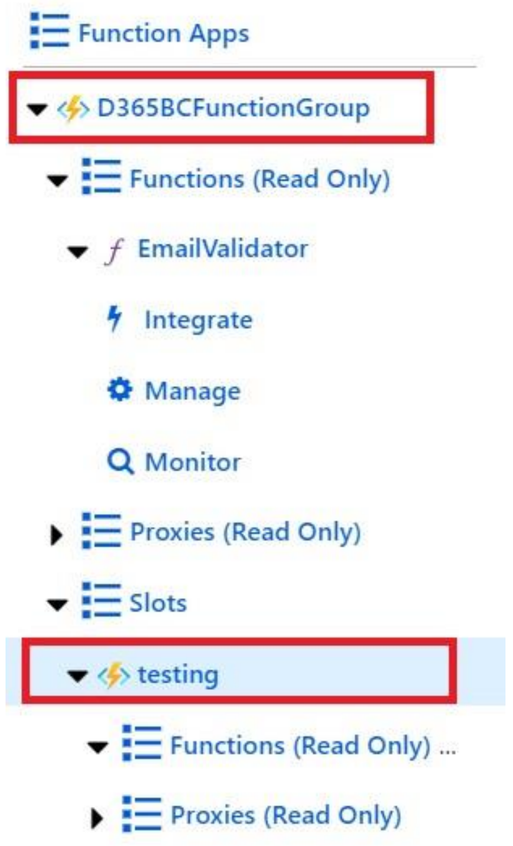


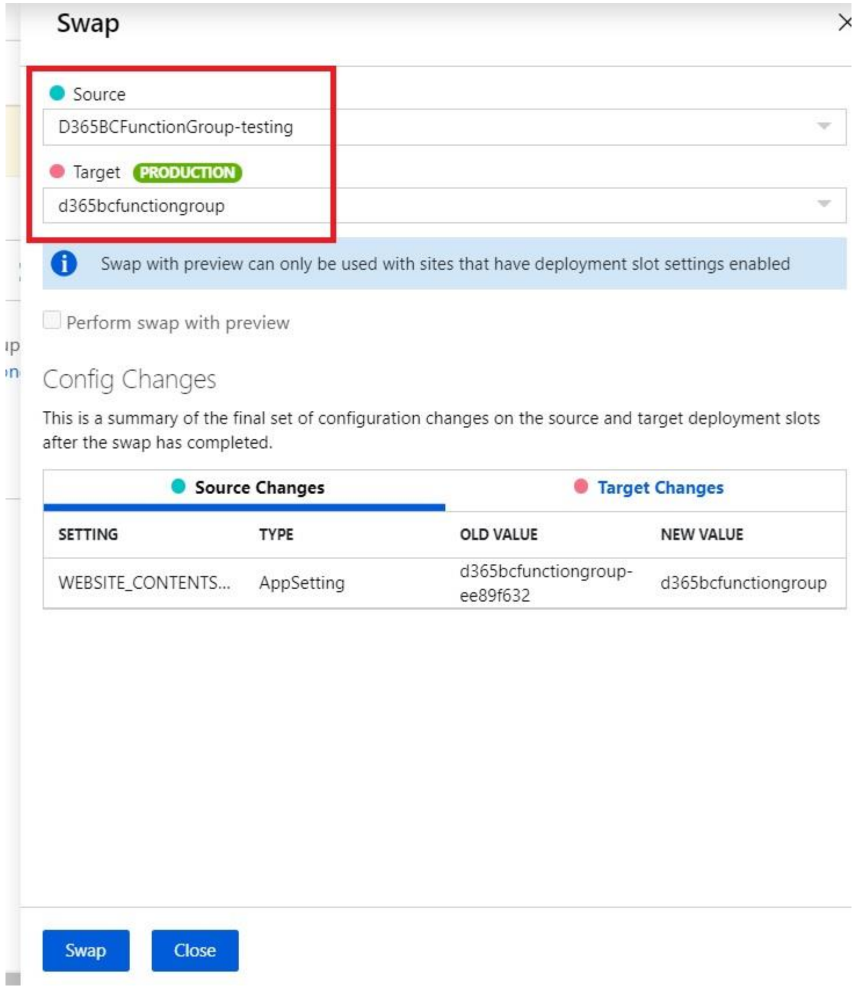


Create a new deployment slot

Deployment slots let you deploy different versions of your function app to different URLs.

Name: testing





Swap

Source: D365BCFunctionGroup-testing

Target PRODUCTION: d365bcfunctiongroup

Swap with preview can only be used with sites that have deployment slot settings enabled

Perform swap with preview

Config Changes

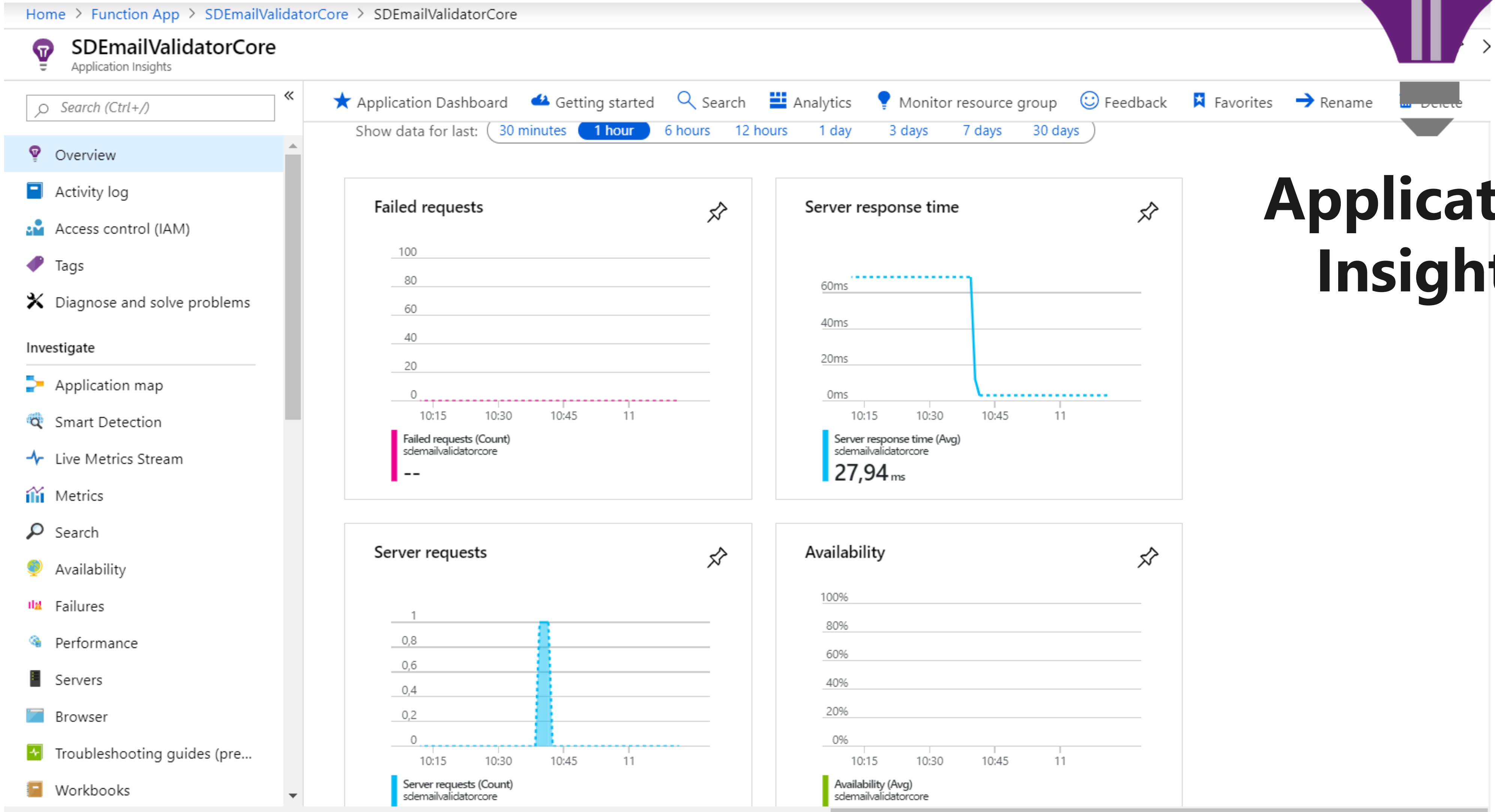
This is a summary of the final set of configuration changes on the source and target deployment slots after the swap has completed.

Source Changes | Target Changes

SETTING	TYPE	OLD VALUE	NEW VALUE
WEBSITE_CONTENTS...	AppSetting	d365bcfunctiongroup-ee89f632	d365bcfunctiongroup

TIP 14: Always monitoring

- Gain real-time observability
- Analyze and debug traces and metrics



Home > Function App > SDEmailValidatorCore > SDEmailValidatorCore

SDEmailValidatorCore
Application Insights

Search (Ctrl+/)

- Overview
- Activity log
- Access control (IAM)
- Tags
- Diagnose and solve problems

Investigate

- Application map
- Smart Detection
- Live Metrics Stream
- Metrics
- Search
- Availability
- Failures
- Performance
- Servers
- Browser
- Troubleshooting guides (pre...
- Workbooks

Application Dashboard | Getting started | Search | Analytics | Monitor resource group | Feedback | Favorites | Rename | Delete

Show data for last: 30 minutes | 1 hour | 6 hours | 12 hours | 1 day | 3 days | 7 days | 30 days

Failed requests
Failed requests (Count)
sdemailvalidatorcore
--

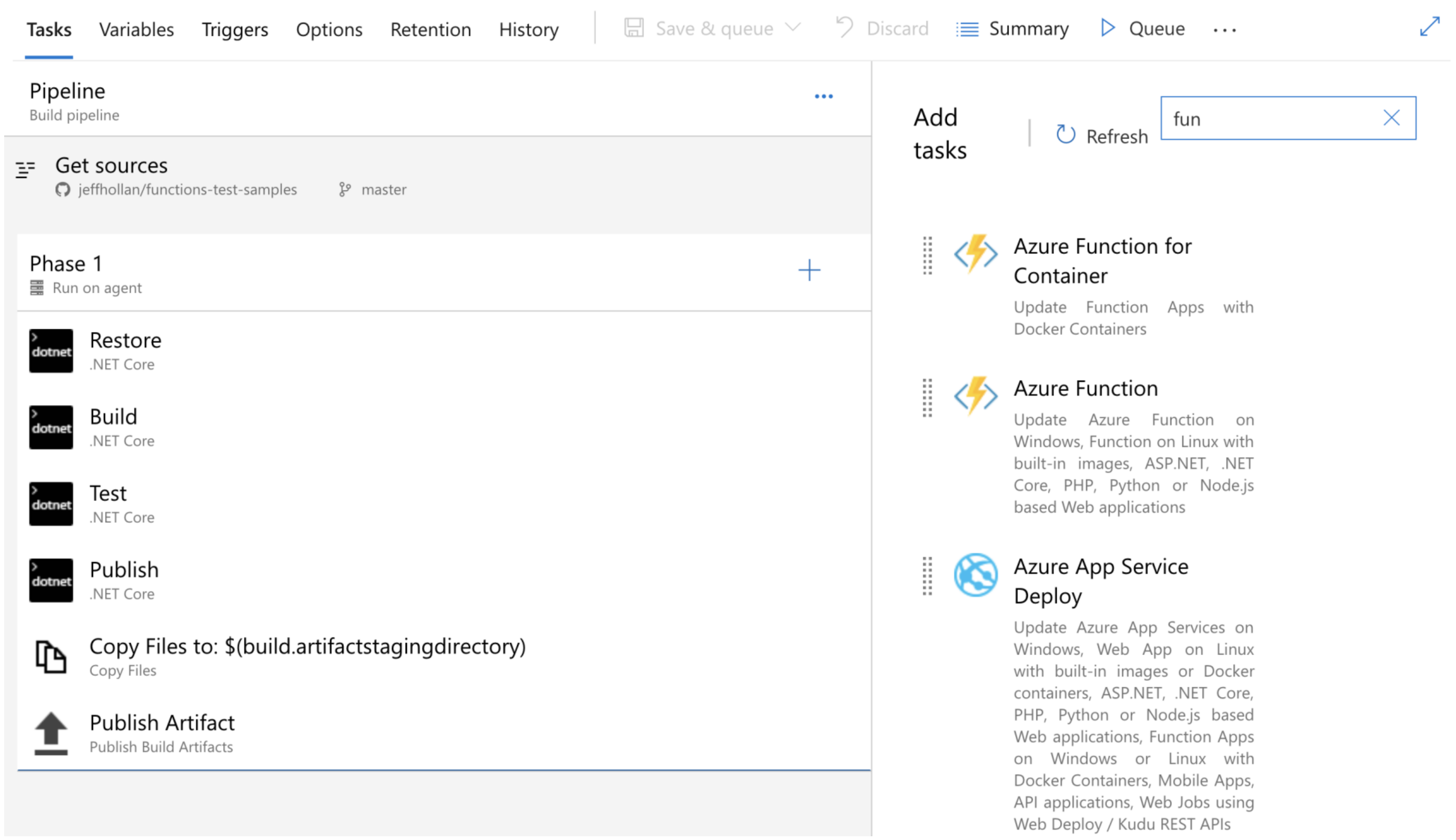
Server response time
Server response time (Avg)
sdemailvalidatorcore
27,94 ms

Server requests
Server requests (Count)
sdemailvalidatorcore

Availability
Availability (Avg)
sdemailvalidatorcore

Application Insights

TIP 15: Azure DevOps for CI/CD



Tasks Variables Triggers Options Retention History | Save & queue Discard Summary Queue ...

Pipeline
Build pipeline

Get sources
jeffhollan/functions-test-samples master

Phase 1
Run on agent

Restore
.NET Core

Build
.NET Core

Test
.NET Core

Publish
.NET Core

Copy Files to: $(build.artifactstagingdirectory)
Copy Files

Publish Artifact
Publish Build Artifacts

Add tasks | Refresh | fun

Azure Function for Container
Update Function Apps with Docker Containers

Azure Function
Update Azure Function on Windows, Function on Linux with built-in images, ASP.NET, .NET Core, PHP, Python or Node.js based Web applications

Azure App Service Deploy
Update Azure App Services on Windows, Web App on Linux with built-in images or Docker containers, ASP.NET, .NET Core, PHP, Python or Node.js based Web applications, Function Apps on Windows or Linux with Docker Containers, Mobile Apps, API applications, Web Jobs using Web Deploy / Kudu REST APIs

Thanks

Questions?

https://github.com/demiliani

@demiliani

https://www.linkedin.com/in/stefano-demiliani/