Azure Functions
Serverless Development Guide

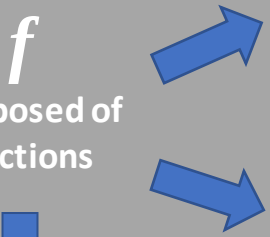
Incoming Events
(File uploaded, message sent, API called)

HTTP

Function apps

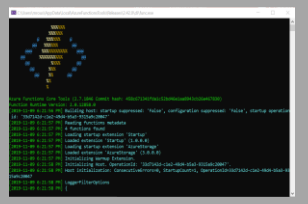
Telemetry sent to
Application Insights

Azure Functions Project



Composed of
functions

Unit Tested



Integration Tested
with Local Runtime

Code committed

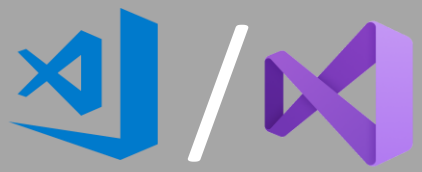
Azure DevOps Repos

Build Definition Triggered

Azure DevOps Pipelines

Deployed to Azure
(Development, Staging, and Production)

Written by Marc Roussy / MarcRoussy.com / @marcroussy



Azure Functions Project

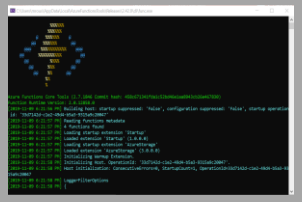
- Serverless is based on event-driven architecture.
- Every function in the project is triggered by a different event.
- Available languages: C#, F#, PowerShell, JavaScript, TypeScript, Java, Python.
- All functions in a project must be in the same language.
- Find out more about the structure of the files and folders in the project.
- Use Visual Studio for .NET-based functions, VS Code for everything else.

Composed of functions *f*

- Functions should follow the single responsibility principle: do only one thing.
- Most useful event triggers are HTTP, Timers, and messaging (Event Hub, Event Grid, Service Bus).
- Input bindings are used in addition to the event trigger to receive additional context from another Azure resource, such as a related record in CosmosDb.
- Output bindings provide data to another service upon function completion. For example, the function sends a message to another process via Service Bus.

Unit Testing

- Unit tests are executed on function triggers or any other class in the project.
- MsTest, nUnit or xUnit can be used to run the unit tests.
- Mocking framework must support .NET Core. Moq and nSubstitute are both great choices.
- Dependency injection can mock a function's event trigger, if needed.



Integration Testing with Local Runtime

- The local runtime simulates Azure's environment on your machine.
- The runtime starts the functions so that you can invoke their HTTP endpoints, process messages from event hubs, execute timers, etc.
- Application Settings, like connection strings, are loaded from `local.settings.json`.
- Runtime configuration, which affects all functions in a project, is loaded from `host.json`

Code committed

This guide uses Azure DevOps for source control, continuous integration and continuous deployment because it's the easiest way to get functions deployed to Azure. The process is similar regardless of the tools you choose to use.

Azure DevOps Repos

- Repo Strategy : Keep projects in the same language in the same repository.
  - Easier to share and restructure code when all similar functions share a repo.
  - Path Filters ensure the correct project is built in a multi-project repository.
  - Avoid storing `local.settings.json` in the repository. It contains secrets and connection strings. Add an entry to the `.gitignore` to remove it
  - Trunk-Based Development is a natural fit for Azure Functions.
    - Frequent integration with others reduces merge conflicts.
    - It's easy to fully test the functions before you push any code to the remote.

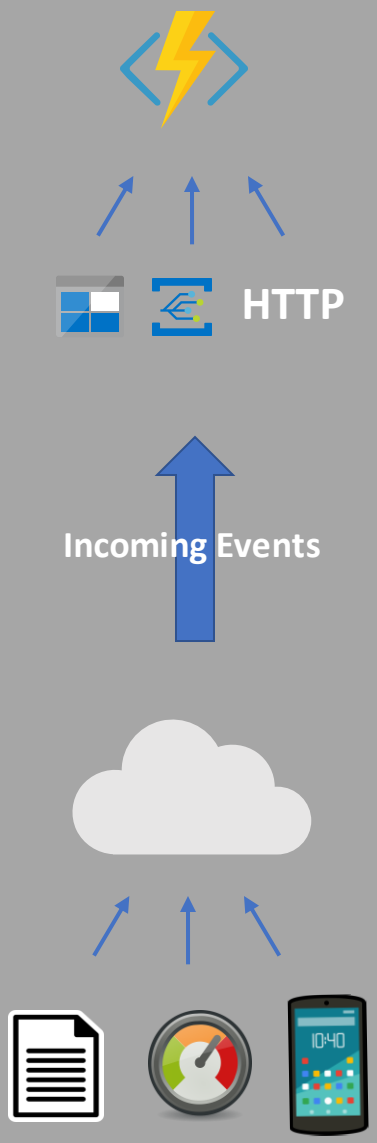
Build Definition Trigger

- The Build and Deployment strategy prepares and releases your code to the cloud.
  - The Build Pipeline is triggered when code is pushed to master.
    - Compiles the code.
    - Runs Unit Tests. The build fails if any tests fail.
    - Creates Artifact for deployment.
  - The Release Pipeline deploys the artifact that was created in the Build Pipeline.
    - Each environment is deployed to in sequence from development to staging to production. The process is the same for each environment:
      - Deploy the artifact to a Function app running in Azure.
      - Set Application Settings. The keys are the same as what's stored in `local.settings.json`, but the values are different.
      - Validate deployment is successful with acceptance tests.
      - Repeat until live in production.

Azure DevOps Pipelines

Deploy to Azure Function apps

Written by Marc Roussy / MarcRoussy.com / @marcroussy



- In the cloud, an Azure Functions project runs within a Function app.
- Each project deploys to its own Function app.
- The Function app is the unit of scale and cost. There are three cost plans:
  - Consumption: Pay as you go, per invocation and execution time in GB-seconds.
  - Premium: Same as consumption, with lower latency, billed per hour.
  - Dedicated: Uses existing App Service instances that are underutilized.
- Start with the Consumption plan, and re-evaluate as your usage increases.
- Functions that run in the Consumption plan are impacted by cold starts.
- Cold starts result in longer execution time since the Function app needs to be loaded into memory before it can run.

- Application Insights monitors the execution of the functions.
- It captures requests, exceptions, and custom events from your code.
- Unlike with other types of applications, Application Insights doesn't provide details on dependencies used by a Function app.

- Function apps for Development, Staging and Production environments expose the same functions with different application settings.
  - Create separate subscriptions and resource groups in Azure to logically separate Function apps across environments.
  - Name Function apps in a pattern like `{environmentname-projectname}`

- Other applications, services, and clients interact with the functions by way of their event triggers, such as:
  - A mobile app that makes HTTP calls to an endpoint.
  - An IoT device that sends messages to an Event Hub.
  - A new document uploaded to a blob.

For related articles, code samples, and more, click here.

Written by Marc Roussy / MarcRoussy.com / @marcroussy