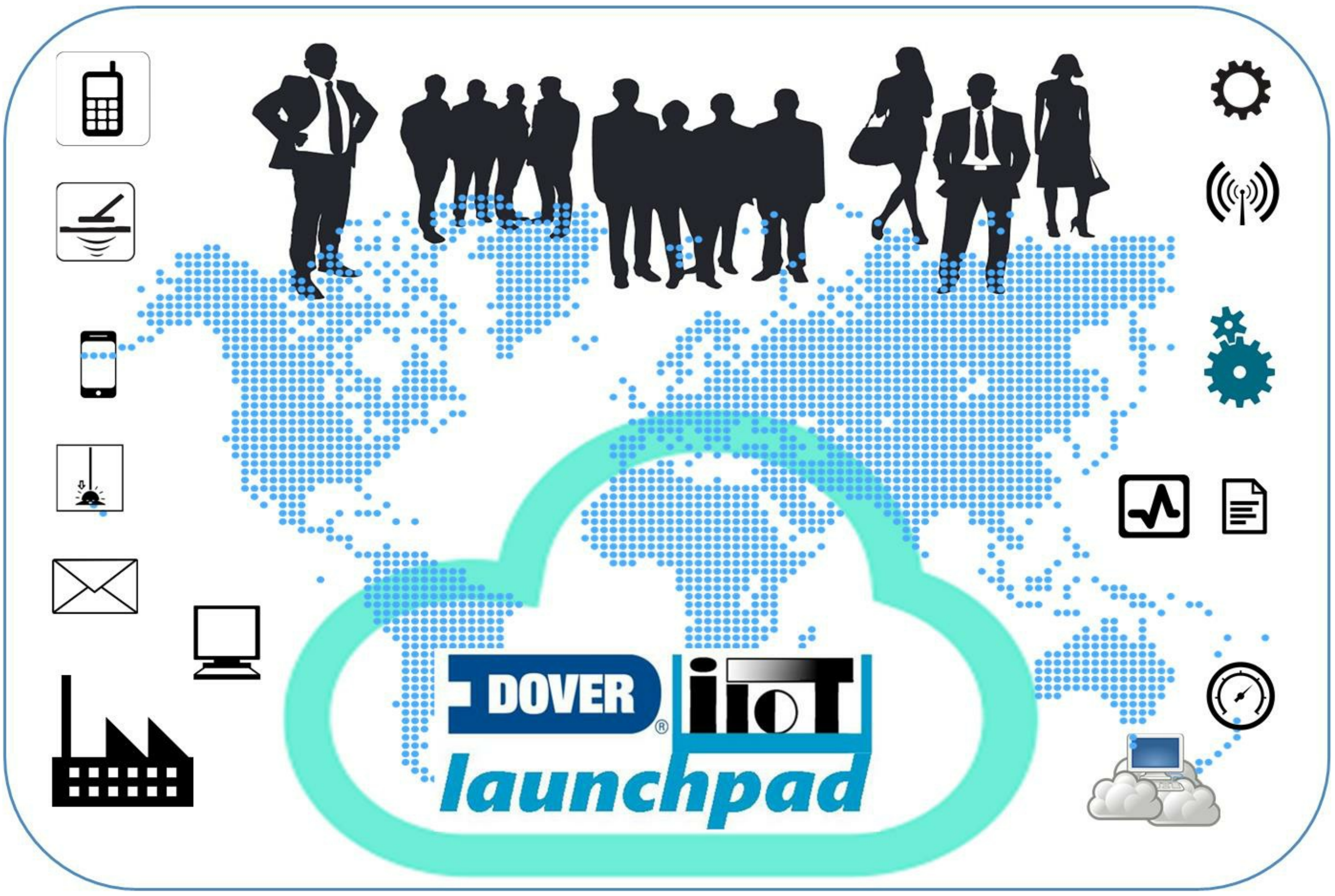


DDS Launchpad IoT Proof Of Concept

This sample project demonstrates the basics of the DDS Launchpad concepts. It will include the fundamental components of an IoT solution using Azure IoT Hub for device message ingress and Azure Service Fabric for device message access and processing.

Setup

1. Install Visual Studio 2017. Any version will do. Make sure you have the following workloads installed
   - Azure development
   - .NET Core cross-platform development
2. Set up your Service Fabric development environment.
3. Create an IoT Hub in Azure or use an existing IoT Hub in your Azure subscription that is not currently being used in a production application. The sample application will run without IoT Hub, but you won't be able to send devices messages through without it. If you're creating a new IoT Hub to run the sample, you can simply use the **F1 - Free** tier under **pricing and scale tier** in the Azure portal.
4. While in Azure IoT Hub make sure to collect the IoT Hub Connection string for later use when configuring the Launchpad applications (.../Shared access policies/iothubowner/Connection string - primary key.)

Deploy

1. Make sure you have a local cluster running
2. Open a PowerShell window and CD to `service-fabric-dotnet-iot\build`
3. Connect to your cluster using `Connect-ServiceFabricCluster`.
4. Run `.\deploy.ps1`
   - The deployment command defaults to a local 5-node cluster. If you're running a different configuration, pass in the name of the publish profile you want to use. For example, for a one-node cluster, use `.\deploy.ps1 -PublishProfileName Local.1Node`.

View application traces

1. Open the solution in Visual Studio.
2. Open the Diagnostics Event viewer: View -> Other Windows -> Diagnostic events
3. In the Diagnostics Event viewer, click the Configuration button with the gear icon, and replace the existing ETW providers with the following ETW providers, then click "Apply."

```
Microsoft-ServiceFabric-Services
Microsoft-Launchpad.Iot.Admin.Application
Microsoft-Launchpad.IoT.Admin.WebService
Microsoft-Launchpad.IoT.EventsProcessor.Application
Microsoft-Launchpad.IoT.EventsProcessor.RouterService
Microsoft.Launchpad.IoT.Insight.Application
Microsoft.Launchpad.IoT.Insigth.DataService
Microsoft.Launchpad.IoT.Insigth.WebService
```

Have fun

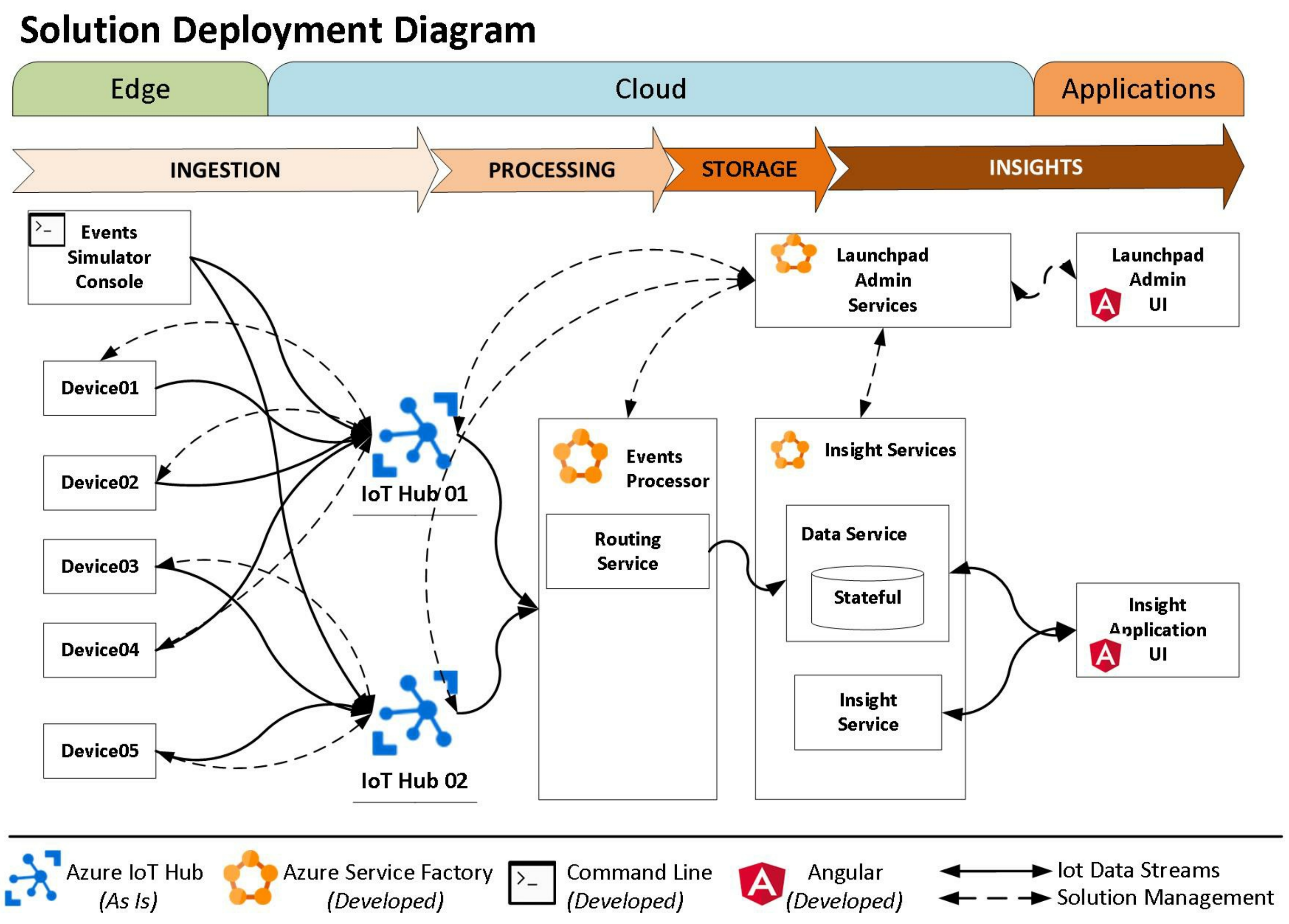
1. Once the application deployment has completed, go to `http://localhost:8081/launchpad/iot` in a web browser to access the admin web UI.
2. Using the admin UI, create an Event Processor application. Give it any name, add the [IoT Hub connection string] collected earlier when doing the setup of the IoT Hub, and click "Add.
3. Using the admin UI, create an Insight application. Give it any name, any number of data service partitions, 1 web service instance if running locally, or -1 if running it Azure, and click "Add."
4. Once the tenant application is created, click the "Web portal" link to see the application's main page. *Note:* It may take a minute for the web portal to become available.
5. Run the simple device emulator that's included with the sample under `dds-launchpad-iot\src\Iot.DeviceEmulator`.
   1. Open then Insighton in Visual Studio 2015.
   2. Set the Iot.DeviceEmulator project as the start-up project and press F5 or ctrl+F5 to run it.
   3. Follow the instructions in the command prompt to register devices with IoT Hub and send messages to the Insight application created in step 3.

Clean up

1. Open a PowerShell window and CD to `dds-launchpad-iot\build`
2. To completely remove the sample from a cluster, run `.\obliterate.ps1`
3. To clean all build output, run `.\clean.cmd`.

Conceptual overview

In this example, the system allows an administrator to add any number of "insight" applications to the solution to consume messages from device field gateways through any number of IoT Hub instances. Customers (tenants) can view their devices and device messages through a Web UI. Messages sent from devices are expected to include a tenant name and device ID for the message ingestion application (Event Processor) to determine which tenant to send the message to.



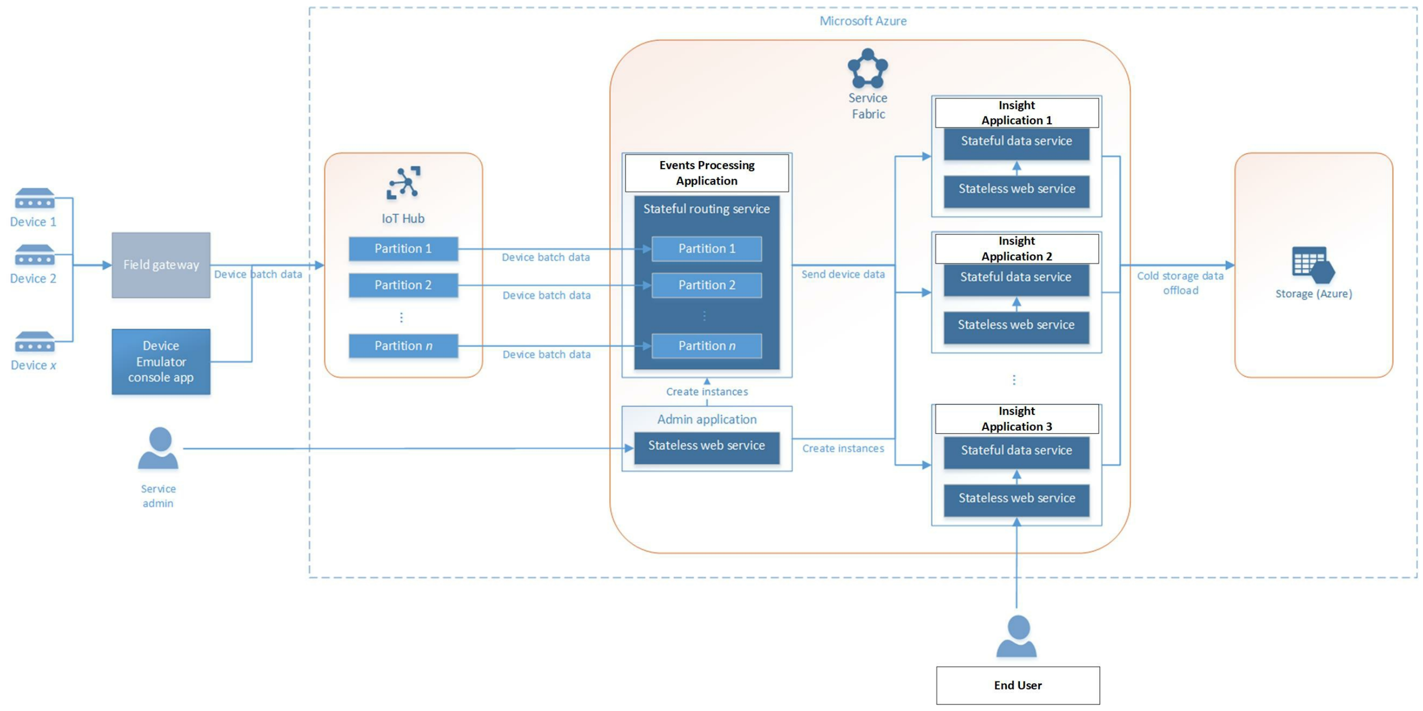
Patterns demonstrated in this sample

- Reading messages from IoT Hub with EventHubReceiver. A partitioned stateful Service Fabric service provides a simple and intuitive way to read from IoT Hub partitions by simply mapping stateful service partitions to IoT Hub partitions one-to-one. This approach does not require any addition coordination or leasing between IoT Hub readers. Service Fabric manages creation, placement, availability, and scaling of the stateful service partitions. A Reliable Collection is used in each partition to keep track of the position, or *offset* in the IoT Hub event stream.
- Multi-solution support using dynamically-created named application instances. An administrative application uses the Service Fabric management APIs to create a new named application instance for each new insight application, so that each application gets their own instance of the message storing and processing service, isolated from other applications.
- Service-to-service communication using HTTP with ASP.NET Core. Services expose HTTP endpoints to communicate with other services.

Architectural overview

This example makes use of several Azure products:

- **IoT Hub** for device management and device message ingress.
- **Service Fabric** hosts the message processing applications.
- **Azure Storage (optional)** can optionally be used for long-term message storage, but is not shown in this example.



Structure

The solution is composed of three separate Service Fabric applications, each with their own set of services:

Admin Application

Used by an adminstrator to create instances of the Event Processor Application and Insight Application.

- **Web Service**: The only service in this application, the web service is a stateless ASP.NET Core service that presents a management UI. API controllers perform Service Fabric management operations using `FabricClient`.

Events Processor Application

An instance of this application connects to an IoT Hub, reads device events, and forwards data to the tenant application identified in the device event message.

- **Router Service**: The only service in this application, the router service is the stateful partitioned service that connects to IoT Hub and forwards device messages to the appropriate tenant application.

Insight Application

A customer (tenant) is the solution owner and will provide "insight" applications to satisfy specific domain related requirements. The solution will have a main customer (tenant), but it would support extending the solution to support additional related organizations such as clients, partners and even consumers. Event data flows from the edge devices into the insight applications for viewing and processing.

- **Data Service**: This is a stateful service that holds the most recent device message for an insight application. This service is partitioned so that solutions with a large number of devices can scale horizontally to meet their requirements.
- **Web Service**: A stateless ASP.NET Core service that presents a UI for the tenant to see devices and their most recent messages.

Deployment

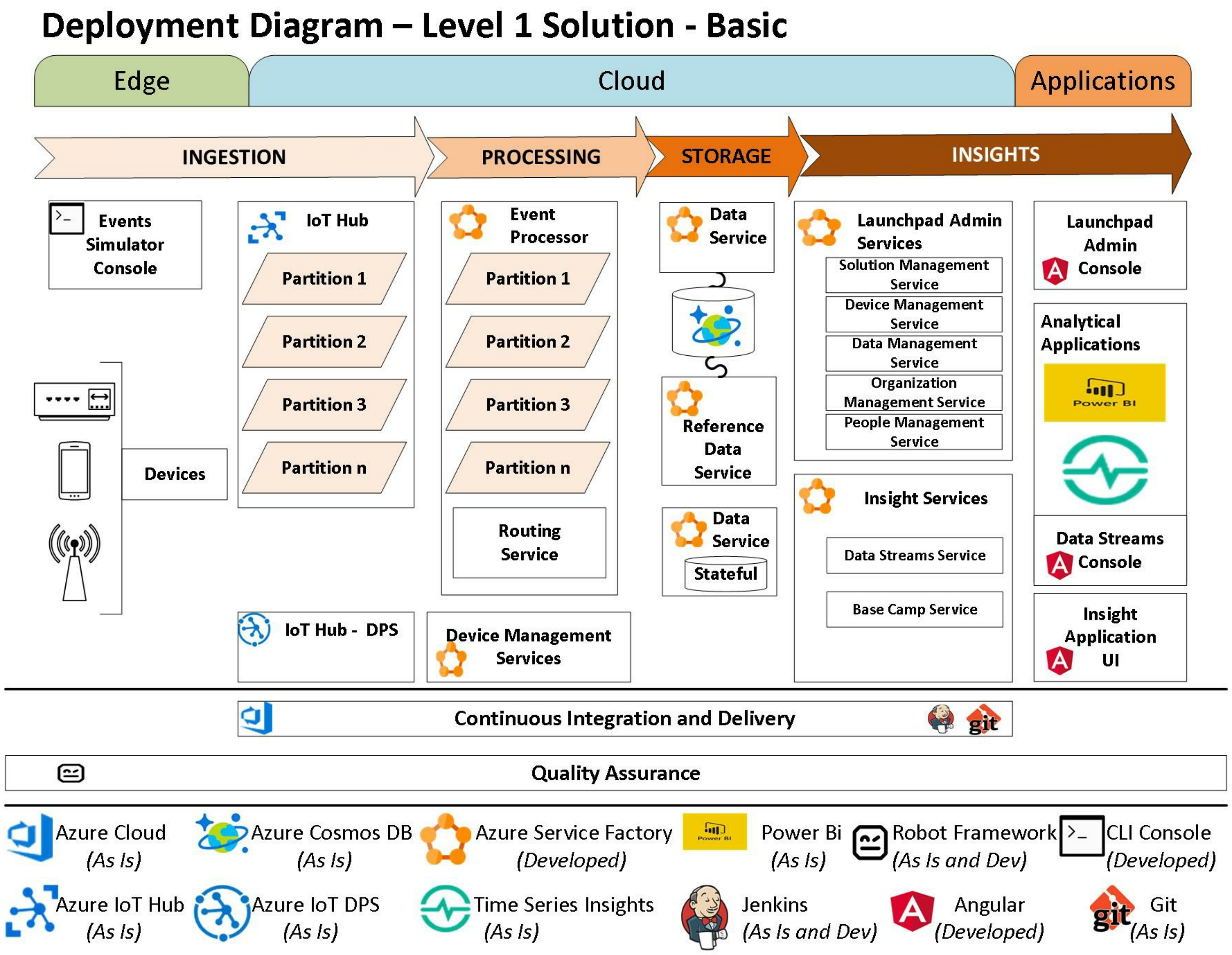
This sample project **does not** create instances of each application during deployment time. The project is designed to create new application instances at runtime through the Admin UI. In order to create application instances at runtime, the application types must be registered at deployment time. A fresh deployment of this sample looks like this:

- LaunchpadIotAdminApplicationType
- LaunchpadIotEventsProcessorApplicationType
- LaunchpadIotInsightApplicationType

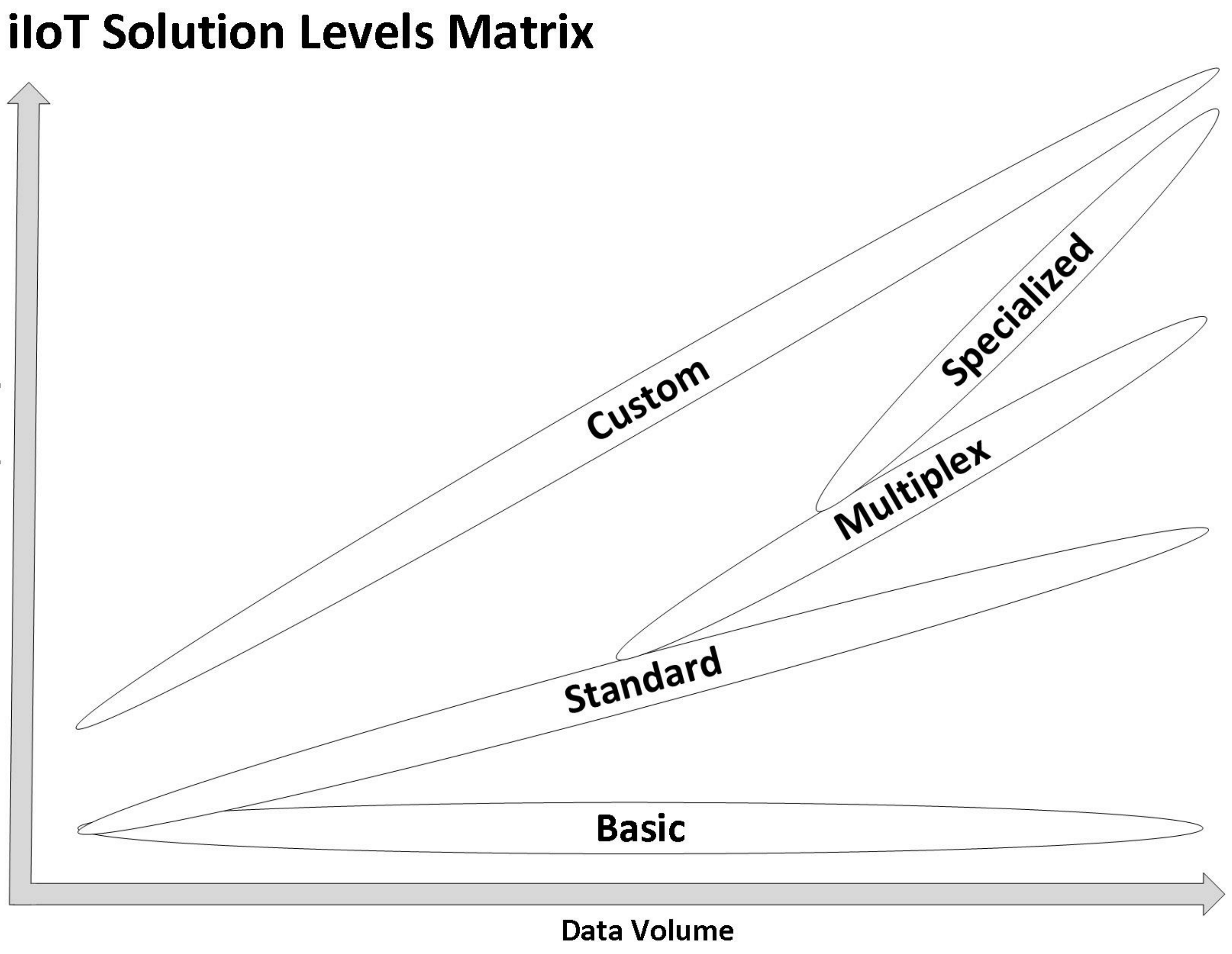
Note that only the LaunchpadIoTAdminApplicationType has a named instance running. The other two application types are registered, but no named application instances are created by default. The web UI in the admin application is used to dynamically create named instances of the other two applications.

Where we go from here

The main purpose of this project is to demonstrate the fundamental patterns we expect to implement as part of the DDS iIoT Launchpad project. The following diagram will give some idea of the scope of the project.



The idea is to use the current POC project as the base for the implementation of the Phase I of the IoT Platform. The phase I should create the artifacts that should support Basic IoT solutions. In the following diagram we have a first exposure to the concept of IoT solution levels.



The concepts behind each of one of those levels is beyond the scope of this document. It suffices to indicate that the "Basic" level escalates well in the dimension of data volumes but is very "economical" in the complexity axis. The idea is provide the basic foundations for the platforms and support the most common use cases.