SERVICE FABRIC

Hands-On

Simon Dale

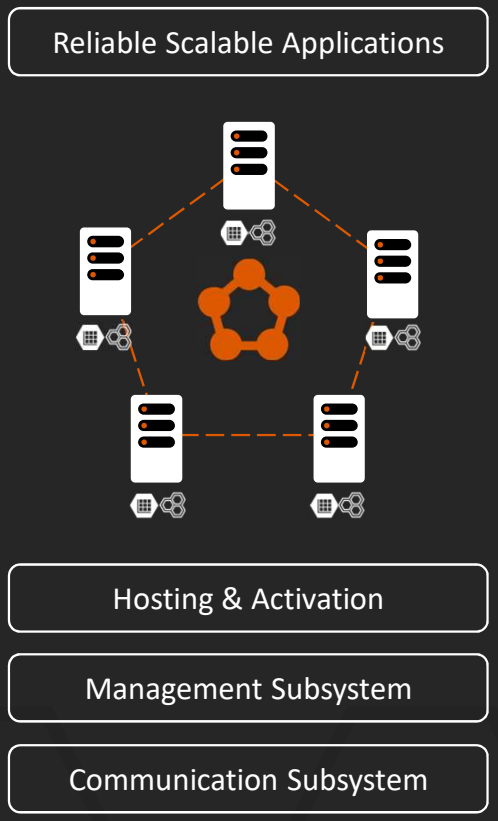
“I'm a Technical Architect with BJSS currently working on Microsoft Azure, Service Fabric and Mixed Reality with the .NET tech stack. I’ve always had an interest for distributed computing, data and producing scalable, performant software.”

bjss

@simondale_
https://github.com/simondale
https://linkedin.com/in/simonjdale

SERVICE FABRIC

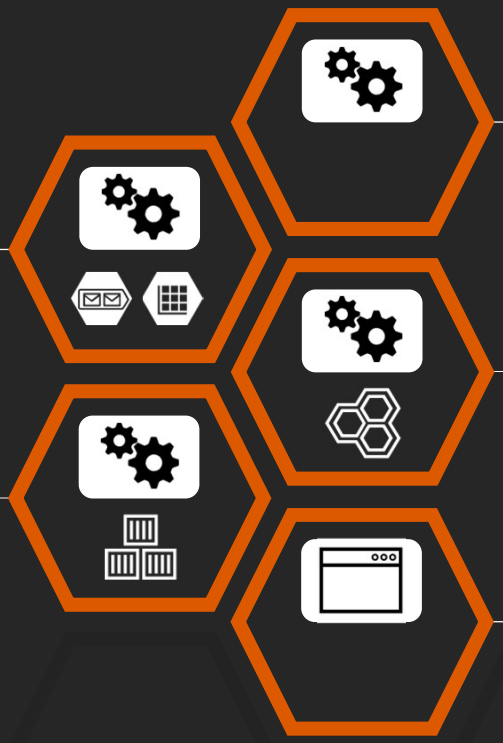
- Distributed Systems Platform
- Scalable, Reliable Microservices and Containers
- Next Generation, Enterprise Class, Cloud Scale
- Application Platform Layer
- Runs on a cluster of machines
- Run anywhere



SERVICE APPROACH

State lives in compute tier
Low latency reads and writes
Partitions for horizontal scale
Data stores for analytics and DR

Stateful



Stateless

State lives within actor instances
Massively scalable, low overhead
Synchronised concurrency
Out of box messaging

Reliable Actors

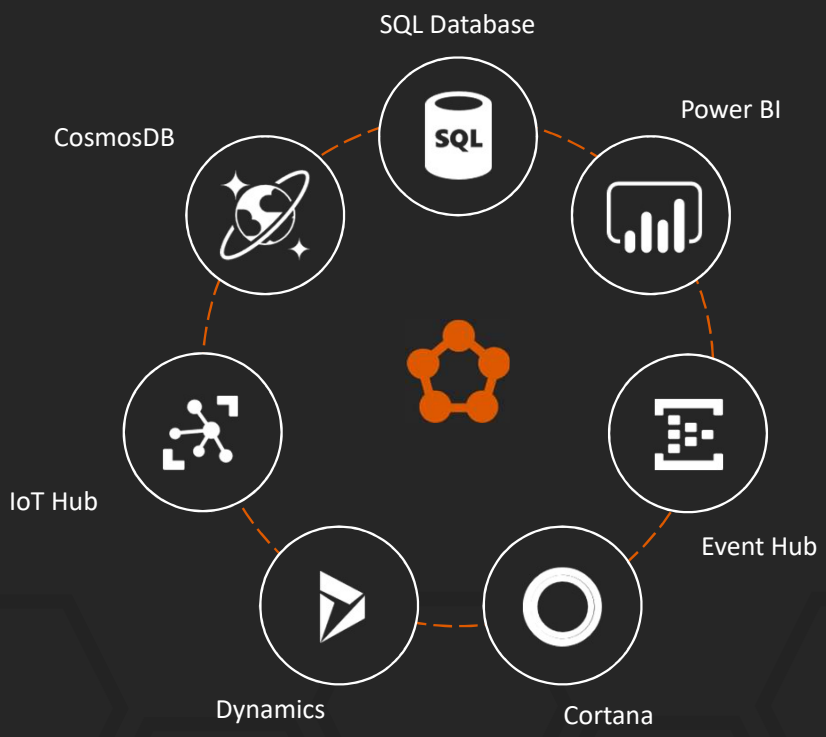
Containers

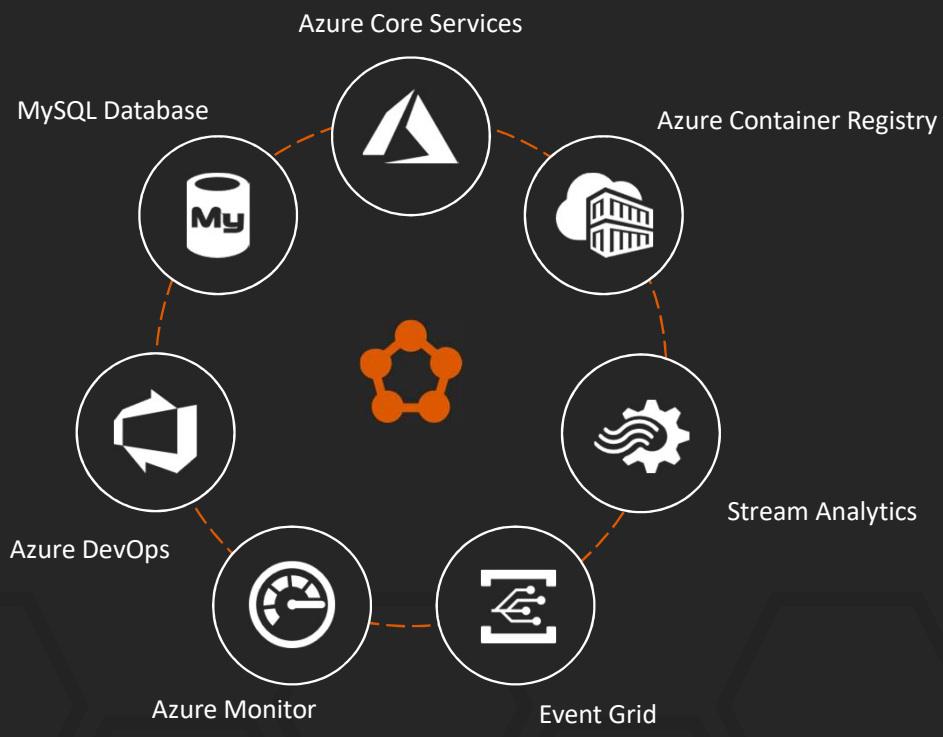
Encapsulate key dependencies
Reuse existing estate
Additional isolation from host
Benefit from placement and scale

Reuse existing applications
Lightweight process execution
Maximise compute density
Benefit from placement and scale

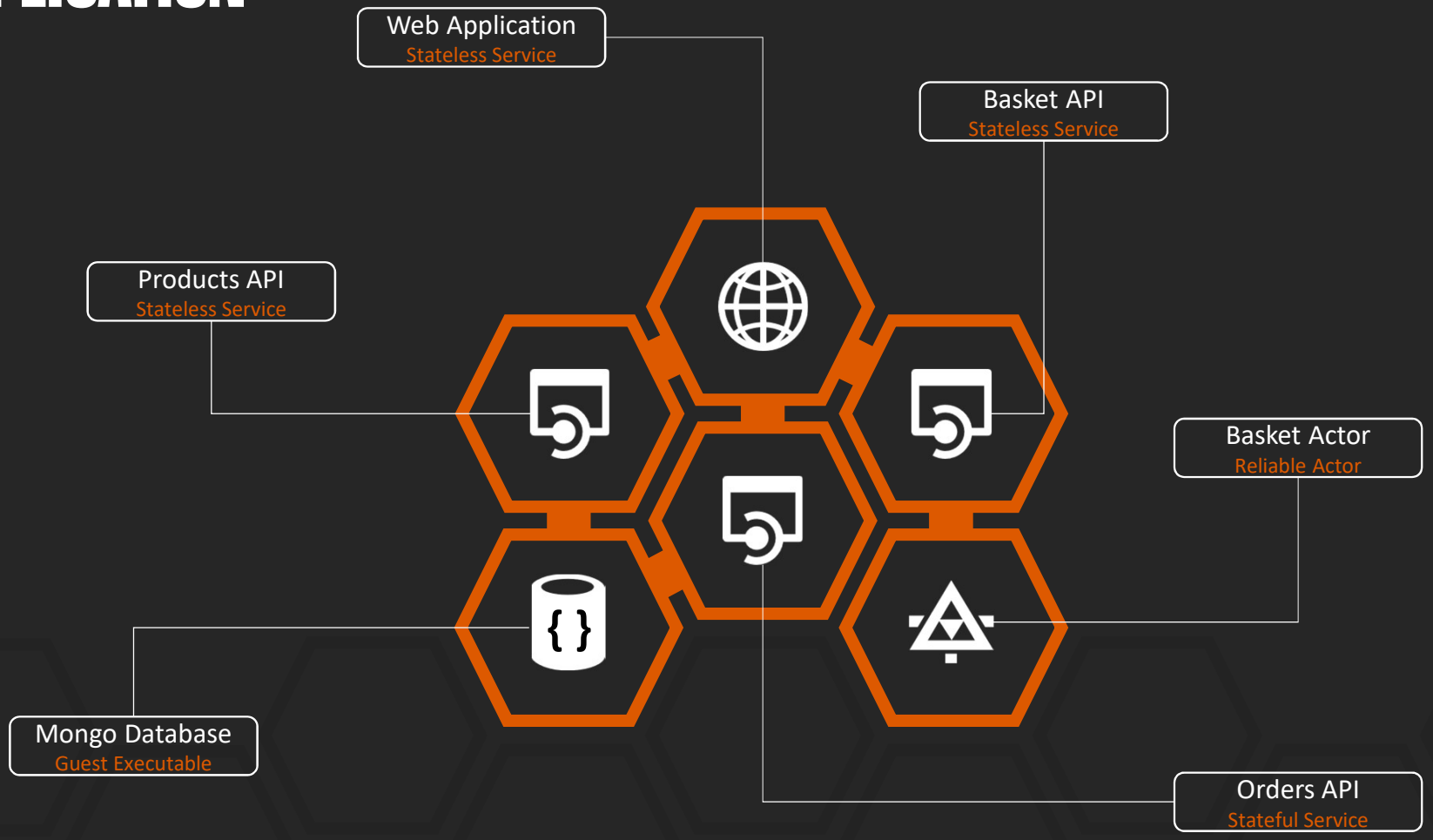
Guest EXE

POWERING AZURE SERVICES

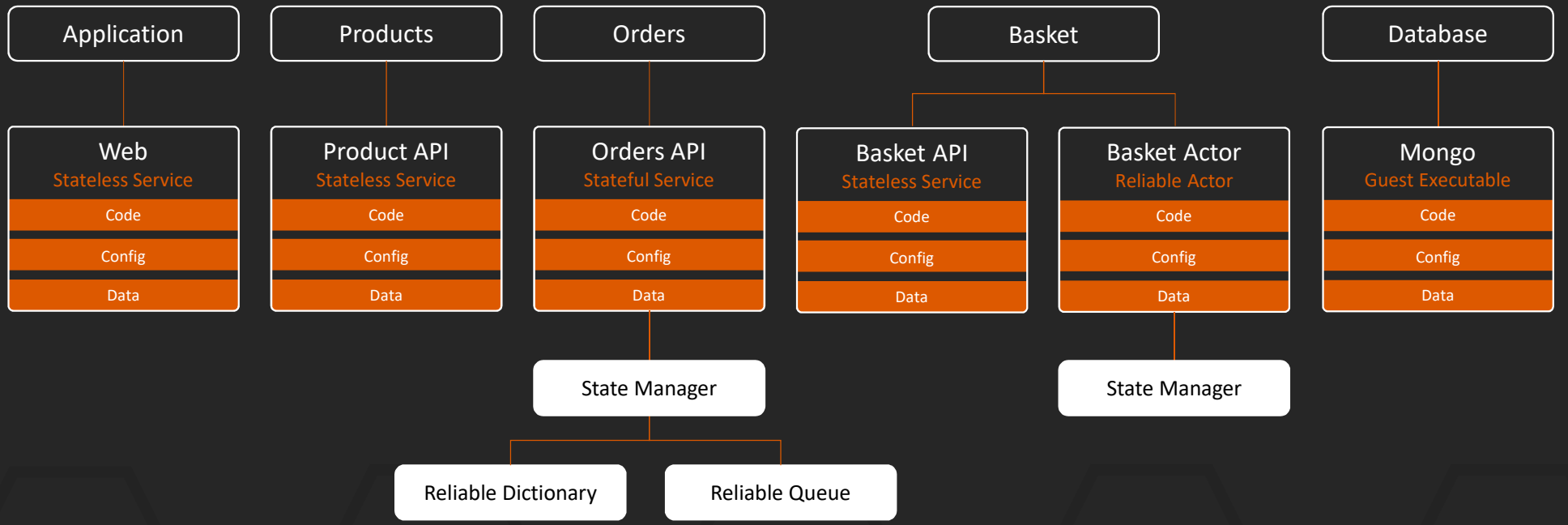




APPLICATION



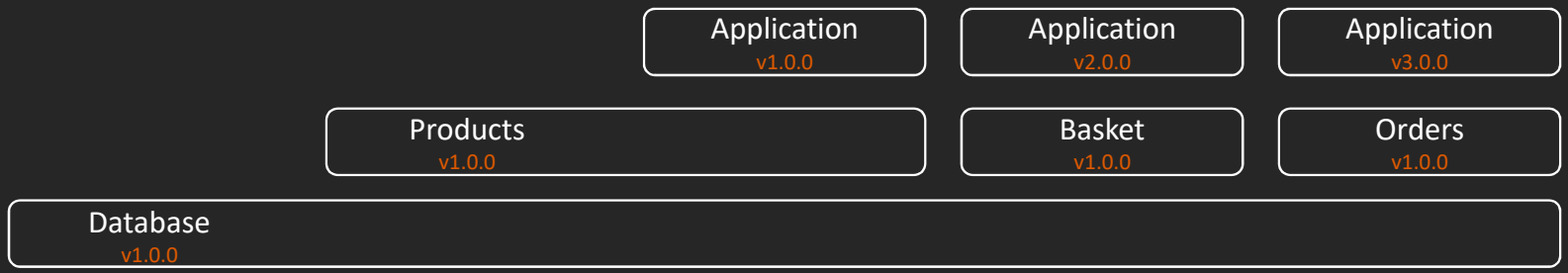
DEPLOYMENT

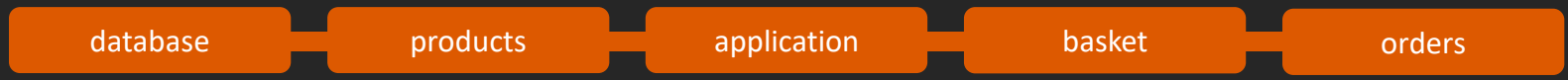

Application
Products
Orders
Basket
Database
Web
Stateless Service
Code
Config
Data
Product API
Stateless Service
Code
Config
Data
Orders API
Stateful Service
Code
Config
Data
Basket API
Stateless Service
Code
Config
Data
Basket Actor
Reliable Actor
Code
Config
Data
Mongo
Guest Executable
Code
Config
Data
State Manager
State Manager
Reliable Dictionary
Reliable Queue

WORKSHOP

Service Fabric

Branches





PREREQUISITES

- Visual Studio Code or Community Edition
  https://visualstudio.microsoft.com
  https://code.visualstudio.com
- Service Fabric SDK
  https://www.microsoft.com/web/handlers/webpi.ashx?command=getinstallerredirect&appid=MicrosoftAzure-ServiceFabric-CoreSDK
- Azure Subscription
  https://portal.azure.com
- Azure DevOps Account
  https://dev.azure.com

DATABASE GUEST EXECUTABLE

```
$ git checkout database
```

- MongoDB
- Guest executable (mongod)
- Test deployment using client (mongo)
- Roadmap to replace this service with Azure PaaS (CosmosDB)

DATABASE GUEST EXECUTABLE

Service Manifest

Defines the Guest Executable service
Specify the entry point program
Optional command line arguments
Working folder
Capture log files and setup log file rotation

```
<EntryPoint>
    <ExeHost>
    <Program>mongod.exe</Program>
    <Arguments>--noauth --ipv6 --bind_ip_all --dbpath=.</Arguments>
    <WorkingFolder>Work</WorkingFolder>
    <ConsoleRedirection FileRetentionCount="5" FileMaxSizeInKb="2048"/>
    </ExeHost>
</EntryPoint>
```

Application Manifest

Define parameters for configurable service scalability

Setup a partitioning strategy
Specify number of instances per partition

The Database application is a singleton with a single instance

```
<Parameters>
    <Parameter Name="Mongo_InstanceCount" DefaultValue="1" />
</Parameters>
```

```
<Service Name="Mongo" ServicePackageActivationMode="ExclusiveProcess">
    <StatelessService ServiceTypeName="MongoType" InstanceCount="[Mongo_InstanceCount]">
        <SingletonPartition />
    </StatelessService>
</Service>
```

STATELESS PRODUCTS WEB API

```
$ git checkout products
```

- ASP.NET MVC WebAPI
- Single controller (Products)
- MongoDB repository for CRUD operations
- Settings containing connection details
- Test using curl or PowerShell

STATELESS PRODUCTS WEB API

Configuration Change

Request configuration packages from code package context
Register for configuration change events
Handle modifications to configuration
In-place updates of executing code

```
public ProductRepository(StatelessServiceContext context)
{
    context.CodePackageActivationContext.ConfigurationPackageModifiedEvent += OnConfigurationPackageModified;
    OnConfigurationPackageModified(this, new PackageModifiedEventArgs<ConfigurationPackage>
    {
        NewPackage = context.CodePackageActivationContext.GetConfigurationPackageObject("Config")
    });
}
```

Service Resolver

Resolve the location of the MongoDB service
Singleton Partition, Singe Instance
Endpoints specified as JSON
Extract hostname and port
Create mongodb://host:port URL

```
private void OnConfigurationPackageModified(object sender, PackageModifiedEventArgs<ConfigurationPackage> e)
{
    var section = e.NewPackage?.Settings?.Sections?["Database"];
    var connectionString = section?.Parameters?["ConnectionString"]?.Value;
    if (connectionString != null)
    {
        if (connectionString.Contains("{application:service}"))
        {
            var application = section.Parameters["Application"].Value;
            var service = section.Parameters["Service"].Value;
            var resolver = ServicePartitionResolver.GetDefault();
            var partition = resolver.ResolveAsync(new Uri($"fabric:/{application}/{service}"), new ServicePartitionKey(), CancellationToken.None).GetAwaiter().GetResult();
            var address = JObject.Parse(partition.Endpoints.Select(ep => ep.Address).First()).SelectToken("Endpoints").ToObject<JObject>().Properties().First()
            .Value.Value<string>();
            connectionString = connectionString.Replace("{application:service}", address);
        }

        var client = new MongoClient(connectionString);
        lock (sync)
        {
            database = client.GetDatabase(section.Parameters["Database"].Value);
            collection = section.Parameters["Collection"].Value;
        }
    }
}
```

STATELESS APPLICATION

```
$ git checkout application
```

- ASP.NET MVC Web Front-End
- Implement a single controller for Products
- Support CRUD operations
- Link to Product API microservice

STATELESS APPLICATION

Application Manifest

Update the Application Manifest file
Specify a well known port for the HTTP endpoint
Set the direction to be incoming requests
Leverage the default NSG rules for Azure Service Fabric

```
<Endpoints>
    <Endpoint Protocol="http" Name="ServiceEndpoint" Type="Input" Port="80" />
</Endpoints>
```

HTTP Communication

Register an HttpClient dependency
HTTP requests to other microservices
Setup Accept and Content-Type headers
JSON serialization of message payloads
Asynchronous calls between services

```
public void ConfigureServices(IServiceCollection services)
{
    services.Configure<CookiePolicyOptions>(options =>
    {
        // This lambda determines whether user consent for non-essential cookies is needed for a given request.
        options.CheckConsentNeeded = context => true;
        options.MinimumSameSitePolicy = SameSiteMode.None;
    });

    services.AddSingleton(new HttpClient());

    services.AddMvc().SetCompatibilityVersion(CompatibilityVersion.Version_2_2);
}
```

BASKET RELIABLE ACTORS

```
$ git checkout basket
```

- ASP.NET MVC WebAPI
- Service Fabric Reliable Actor
- Calls to API request Actor Proxy and issue requests
- Reliable actors maintain state linked to user session
- Expiry timeout for actors
- Update Web Application to add items to Basket

BASKET RELIABLE ACTORS

Actor Proxies

Identify Actors using an ActorID (string, long or GUID)
Request an ActorProxy from a service type and ID
Communicate using well defined interface
Serializable data types

```
[HttpGet("{id}")]
public async Task<ActionResult<Product>> Get(Guid id)
{
    try
    {
        var actorId = new ActorId($"{id:N}");
        var actor = ActorProxy.Create<IBasketActor>(actorId, new Uri("fabric:/Basket/BasketActorService"));
        return Ok(await actor.GetProductsInBasket(CancellationToken.None));
    }
    catch (Exception e)
    {
        var response = Json(new
        {
            e.Message
        });
        response.StatusCode = 500;
        return response;
    }
}
```

Activation / Deactivation

Receive notifications when Actors activate or deactivate
Use to detect disaster recovery scenarios
Keep hot data in memory
Persist cold data to traditional storage

```
protected override Task OnActivateAsync()
{
    ActorEventSource.Current.ActorMessage(this, "Actor activated.");
    return StateManager.TryAddStateAsync(StateName, new List<Product>());
}

protected override Task OnDeactivateAsync()
{
    ActorEventSource.Current.ActorMessage(this, "Actor deactivated.");
    return Task.CompletedTask;
}
```

STATEFUL ORDER SERVICE

```
$ git checkout order-processing
```

- ASP.NET MVC WebAPI
- Stateful Service
- Support competing consumers through Reliable Queue
- Persist order details and statistics to MongoDB
- Maintain current statistics in Reliable Dictionary

STATEFUL ORDER SERVICE

Reliable Queue

Highly available, low-latency buffering of incoming requests
Transactional and ordering guarantees
Failover to warm replicas
Register for disaster recovery events and persist to storage

```
var state = await stateManager.GetOrAddAsync<IReliableConcurrentQueue<Order>>(StateName);

using (var tx = stateManager.CreateTransaction())
{
    await state.EnqueueAsync(tx, order);
    await tx.CommitAsync();
    return Ok();
}
```

Reliable Dictionary

High availability, low-latency, indexed storage
Replicated to a quorum of nodes
Transactional access
Immutable data in storage, update whole object graph

```
var statistics = await stateManager.TryGetAsync<IReliableDictionary2<string, string>>(StatisticsName);

using (var tx = stateManager.CreateTransaction())
{
    var value = await statistics.Value.TryGetValueAsync(tx, id);
    if (!value.HasValue)
    {
        return NotFound();
    }

    var stats = JsonConvert.DeserializeObject<IEnumerable<Statistics>>(value.Value);
    return Ok(stats);
}
```

BUILD IN AZURE

```
$ . src/scripts/deploy-azure.sh
```

- Fully Managed Cluster
- Azure Portal Integration
- Azure Resource Manager
- Auto-scaling
- Integration with Azure Infrastructure

BUILD IN AZURE

Define Variables

Cluster name must be globally unique (e.g. date suffix)
Support for Linux or Windows OS
Different VM SKU available for different workloads
Configurable cluster size
Securely set a VM password

```
: ${CLUSTER_NAME=sfw`date +%Y%m%d%S`}
: ${RESOURCE_GROUP=$CLUSTER_NAME}
: ${LOCATION="westeurope"}
: ${VM_OS="WindowsServer2016Datacenter"}
: ${VM_SKU="Standard_D4_v3"}
: ${CLUSTER_SIZE="5"}
: ${VM_PASSWORD=""}
```

Deploy Cluster

AZCLI will create:

- Service Fabric Cluster
- KeyVault for certificates and secrets
- Virtual Machine Scale Set for Auto-Scale
- Virtual Network (including IP, VNET, LB and NSG)

```
# create the service fabric cluster
az sf cluster create \
    --name $CLUSTER_NAME \
    --resource-group $RESOURCE_GROUP \
    --location $LOCATION \
    --vm-sku $VM_SKU \
    --vm-os $VM_OS \
    --vm-password ${VM_PASSWORD} \
    --certificate-subject-name ${CLUSTER_NAME} \
    --cluster-size ${CLUSTER_SIZE}
```

AZURE DEVOPS INTEGRATION

Login to https://dev.azure.com
Create a new team project
Ensure git source control provider

Create a repository
Import existing code from github
https://github.com/simondale/service-fabric-workshop

Create build pipeline
Select Service Fabric Build template
Add task for each application in repository
Enable continuous integration

Create release pipeline for the build
Select Service Fabric Deploy template
Add task for each artefact in drop
Configure variables and parameter overrides

LOOKING BACK

- What/why/how of Service Fabric
- Service types and deployment
- Built a sample application
- Pipelines with Azure DevOps
- Continuous integration

WHAT'S NEXT

- Azure Containers @dotnetnotts
- What / Why / How
- Docker, DockerHub
- PaaS services with Azure
- Continuous Integration
- Service Fabric Mesh

https://github.com/simondale/azure-containers