Service Fabric – Internals

Microsoft Services

Conditions and Terms of Use

Microsoft Confidential

This training package is proprietary and confidential, and is intended only for uses described in the training materials. Content and software is provided to you under a Non-Disclosure Agreement and cannot be distributed. Copying or disclosing all or any portion of the content and/or software included in such packages is strictly prohibited.

The contents of this package are for informational and training purposes only and are provided "as is" without warranty of any kind, whether express or implied, including but not limited to the implied warranties of merchantability, fitness for a particular purpose, and non-infringement.

Training package content, including URLs and other Internet website references, is subject to change without notice. Because Microsoft must respond to changing market conditions, the content should not be interpreted to be a commitment on the part of Microsoft, and Microsoft cannot guarantee the accuracy of any information presented after the date of publication. Unless otherwise noted, the companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted herein are fictitious, and no association with any real company, organization, product, domain name, e-mail address, logo, person, place, or event is intended or should be inferred.

Copyright and Trademarks

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Microsoft Corporation.

For more information, see **Use of Microsoft Copyrighted Content** at
https://www.microsoft.com/en-us/legal/intellectualproperty/permissions/default.aspx

Microsoft®, Internet Explorer®, Outlook®, SkyDrive®, Windows Vista®, Zune®, Xbox 360®, DirectX®, Windows Server® and Windows® are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. Other Microsoft products mentioned herein may be either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. All other trademarks are property of their respective owners.

Agenda

- Service Fabric Internals
- Resource Balancing
- Endpoints and Service Discovery

Service Fabric – Internals

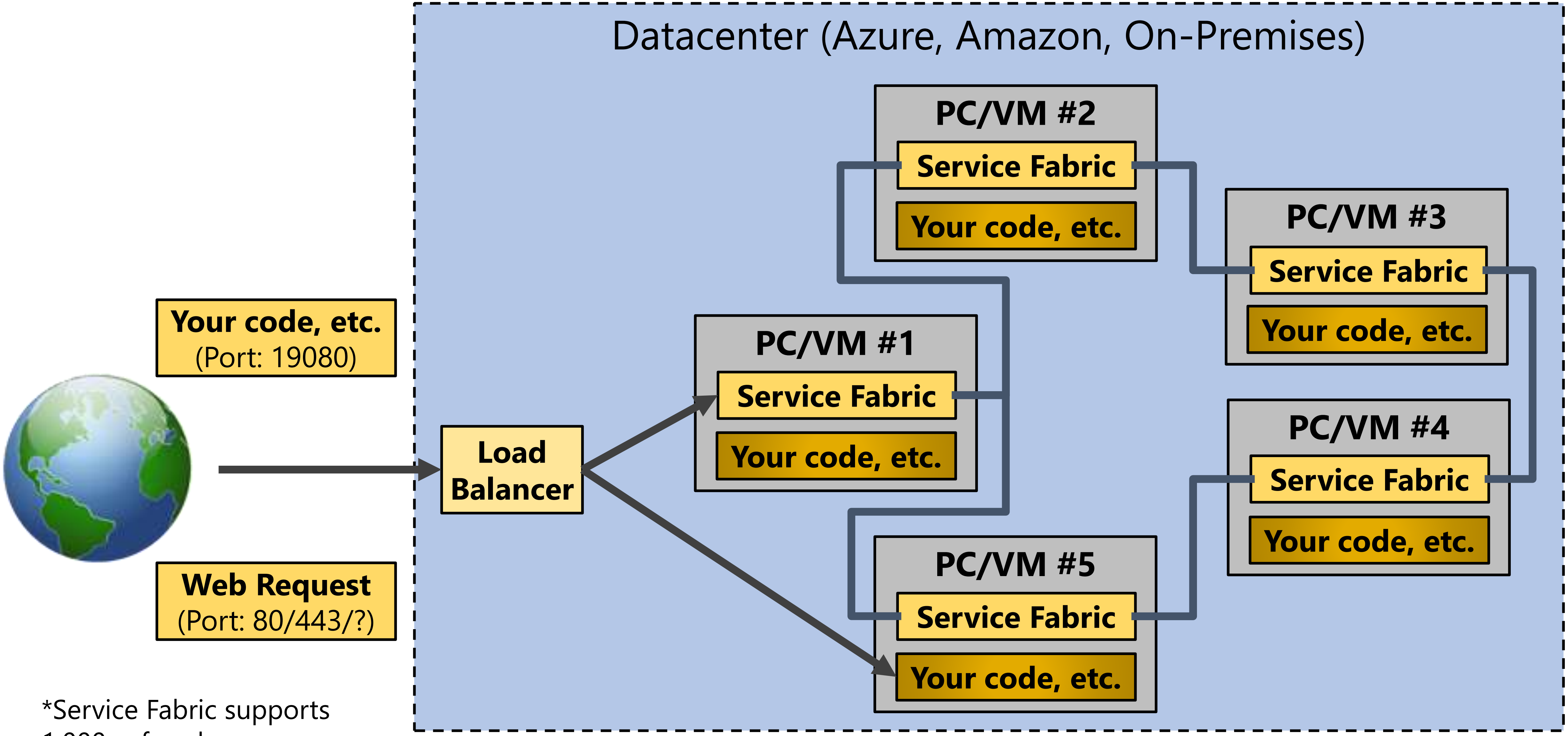
Service Fabric Internals

Microsoft Services

Service Fabric Environments

- Azure
- AzureStack
- On-Premises – Standalone Windows Server
- Other public clouds ~ Standalone Windows Server
- OneBox (local development cluster)

Service Fabric Cluster with 5 Nodes



Datacenter (Azure, Amazon, On-Premises)

Your code, etc.
(Port: 19080)

Web Request
(Port: 80/443/?)

Load Balancer

PC/VM #1
Service Fabric
Your code, etc.

PC/VM #2
Service Fabric
Your code, etc.

PC/VM #3
Service Fabric
Your code, etc.

PC/VM #4
Service Fabric
Your code, etc.

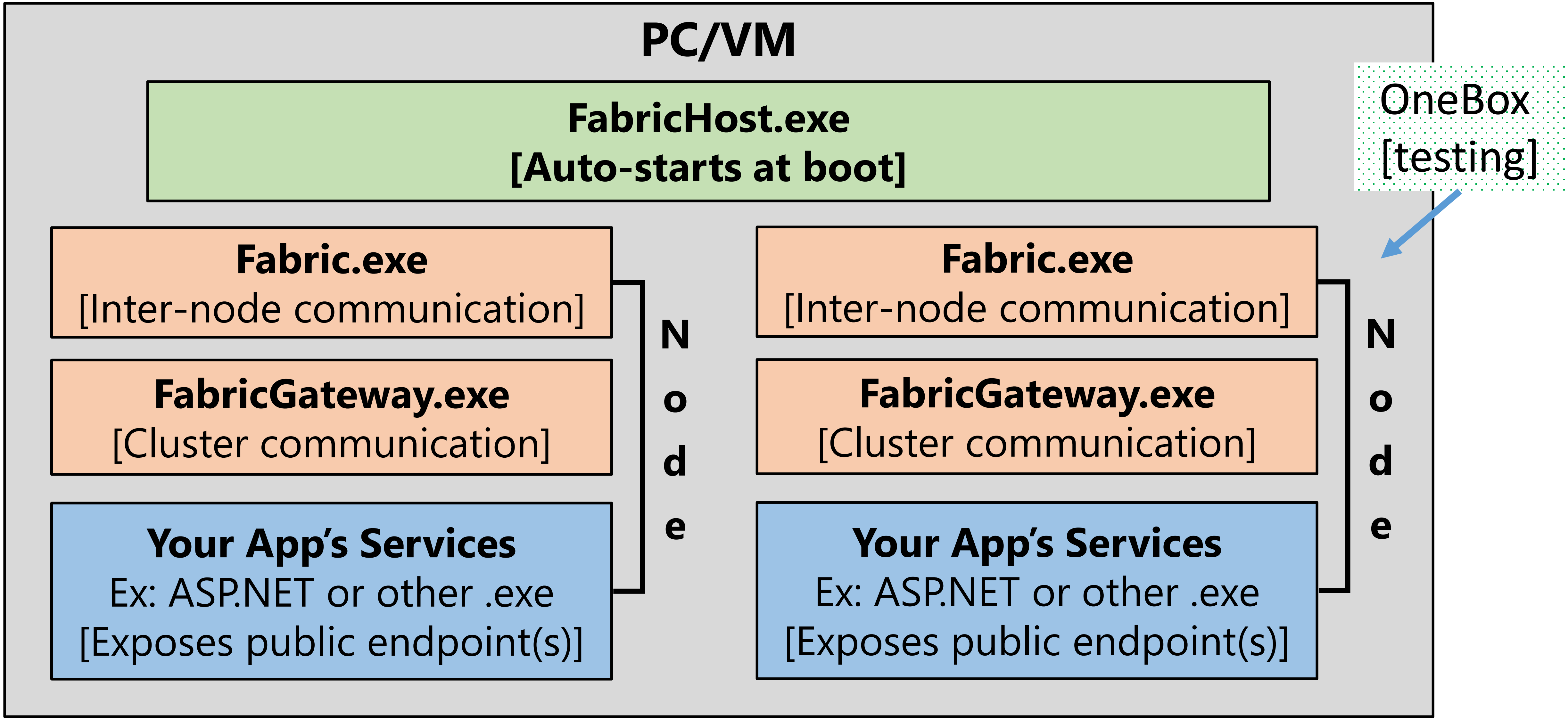
PC/VM #5
Service Fabric
Your code, etc.

*Service Fabric supports
1,000s of nodes

Service Fabric’s Infrastructure Services

Cluster Resource Manager	Ports 19080 [REST] and 19000 [TCP] Performs cluster REST and PowerShell/FabricClient operations
Failover Manager	Rebalances resources as nodes come/go
Naming Service	Maps service instances to endpoints
Image Store (not on OneBox)	Contains your Application packages
Upgrade Services (Azure Only)	Coordinates upgrading Service Fabric itself with Azure’s Service Fabric Resource Provider
Fault Analysis	Let’s you inject faults to test your services

Node Processes



PC/VM

FabricHost.exe
[Auto-starts at boot]

OneBox
[testing]

Fabric.exe
[Inter-node communication]

FabricGateway.exe
[Cluster communication]

Your App's Services
Ex: ASP.NET or other .exe
[Exposes public endpoint(s)]

Node

Fabric.exe
[Inter-node communication]

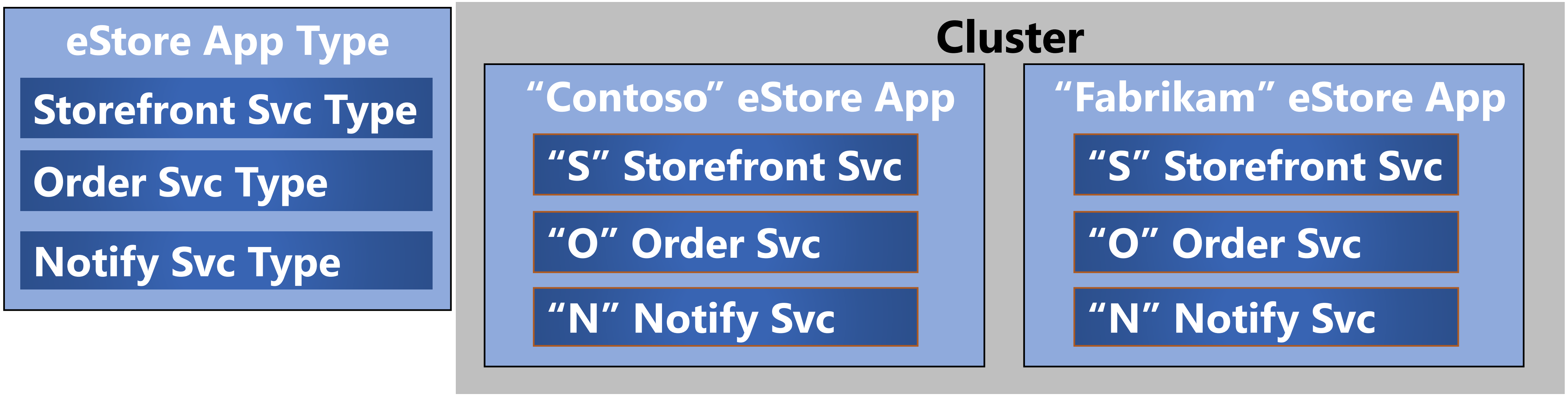
FabricGateway.exe
[Cluster communication]

Your App's Services
Ex: ASP.NET or other .exe
[Exposes public endpoint(s)]

Node

Defining Application Types and Service Types

- An application is a collection of services
    - In Service Fabric terms, we call these *application types* and *service types*
    - An *application type* is a collection of *service types*



eStore App Type

Storefront Svc Type

Order Svc Type

Notify Svc Type

Cluster

“Contoso” eStore App

“S” Storefront Svc

“O” Order Svc

“N” Notify Svc

“Fabrikam” eStore App

“S” Storefront Svc

“O” Order Svc

“N” Notify Svc

- Package the application types and services
    - A package is a directory with an XML manifest file

Application Pkg Dir and its Manifest XML File

- WebServerAppTypePkg
  - ApplicationManifest.xml
  - ***WebServerServicePkg***
    - ServiceManifest.xml
    - CodePkg
      - WebServer.exe
      - WebServer.pdb

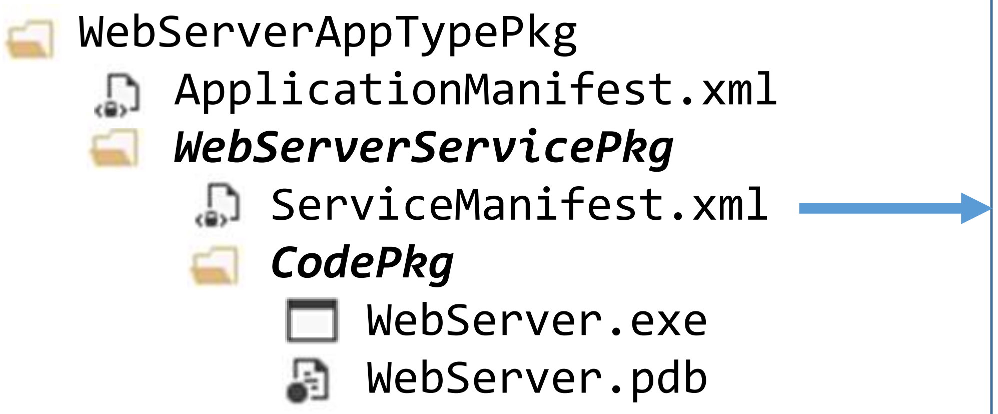
```xml
<ApplicationManifest
   ApplicationTypeName="WebServerAppType"
   ApplicationTypeVersion="1.0" ...>
   <ServiceManifestImport>

      <ServiceManifestRef
         ServiceManifestName="WebServerServicePkg"
         ServiceManifestVersion="1.0" ... />

      ...

   </ServiceManifestImport>
</ApplicationManifest>
```

Service Pkg Directory and its Manifest XML File



```
WebServerAppTypePkg
    ApplicationManifest.xml
    WebServerServicePkg
        ServiceManifest.xml
        CodePkg
            WebServer.exe
            WebServer.pdb
```

```
<ServiceManifest Name="WebServerServicePkg"
                 Version="1.0">
  <ServiceTypes>
    <StatelessServiceType
      ServiceTypeName="WebServerServiceType" ... >
    </StatelessServiceType>
  </ServiceTypes>
  <CodePackage Name="CodePkg" Version="1.0">
    <EntryPoint> <ExeHost>
      <Program>WebServer.exe</Program>
    </ExeHost> </EntryPoint>
  </CodePackage>
  <Resources> <Endpoints>
      <Endpoint Name="WebServerTypeEndpoint"
       Type="Input" Protocol="http" Port="8080" />
    </Endpoints> </Resources>
</ServiceManifest>
```

ServiceManifest.xml - <EntryPoint>

- Defined in ServiceManifest
- Used to specify how to launch the service
- **ExeHost** element specifies the executable that should be used to start the service

- Program specifies the name of the executable that should run in order to start the service.
- Arguments specifies the arguments that should be passed to the executable. It can be a list of parameters with arguments

ServiceManifest.xml - Console redirection

- **ConsoleRedirection** - redirect console output (both stdout and stderr) to a working directory so they can be used to verify that there are no errors during the setup or execution of the application in the Service Fabric cluster
  - **FileRetentionCount** determines how many files are saved in the working directory. A value of 5, for instance, means that the log files for the previous five executions are stored in the working directory
  - **FileMaxSizeInKb** specifies the max size of the log files
- Look on the primary node in the Service Fabric Explorer to determine the location of the log files

Applications, Hosts and Activation

- Applications are packages, copied to the cluster (System:Image Store Service), then registered as an ApplicationType and ApplicationTypeVersion with Service Fabric
- An application instance is based on ApplicationType and ApplicationTypeVersion and defines process isolation boundaries, while a partition defines data isolation boundaries
- The ApplicationManifest's <DefaultServices> are activated with every app instance
- VS.NET tooling creates 1 App Instance when you F5 or Publish
- If you want multiple application instances, create a new application name using the same ApplicationType and ApplicationTypeVersion
- Services are defined as a ServiceType with ServiceTypeVersion in a service package that also has Code, Config and Data packages (each package is versioned)

Creating Apps, Services, Partitions, and Instances

Registered and provisioned

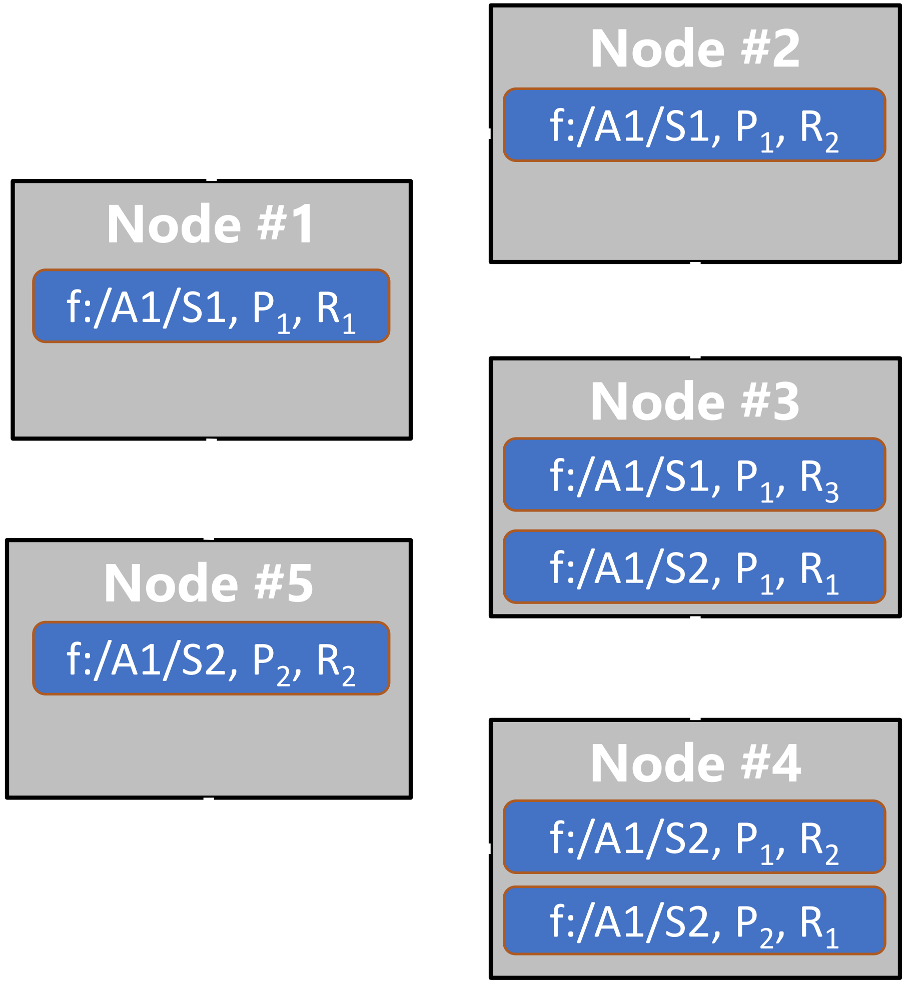
App type="A" with Service type="S"

Create 1 named app

App Type	App Version	App Name
"A"	1.0	fabric:/A1

Creates 2 named services

App Name	Service Type	Service Name	# Partitions	# Replicas
fabric:/A1	"S"	fabric:/A1/S1	1	3
fabric:/A1	"S"	fabric:/A1/S2	2	2



Node #2

f:/A1/S1, P_1, R_2

Node #1

f:/A1/S1, P_1, R_1

Node #3

f:/A1/S1, P_1, R_3

f:/A1/S2, P_1, R_1

Node #5

f:/A1/S2, P_2, R_2

Node #4

f:/A1/S2, P_1, R_2

f:/A1/S2, P_2, R_1

NOTE: When using Service Fabric programming models, instances from same named app/service are in the same process

Auto-scale Service Fabric clusters

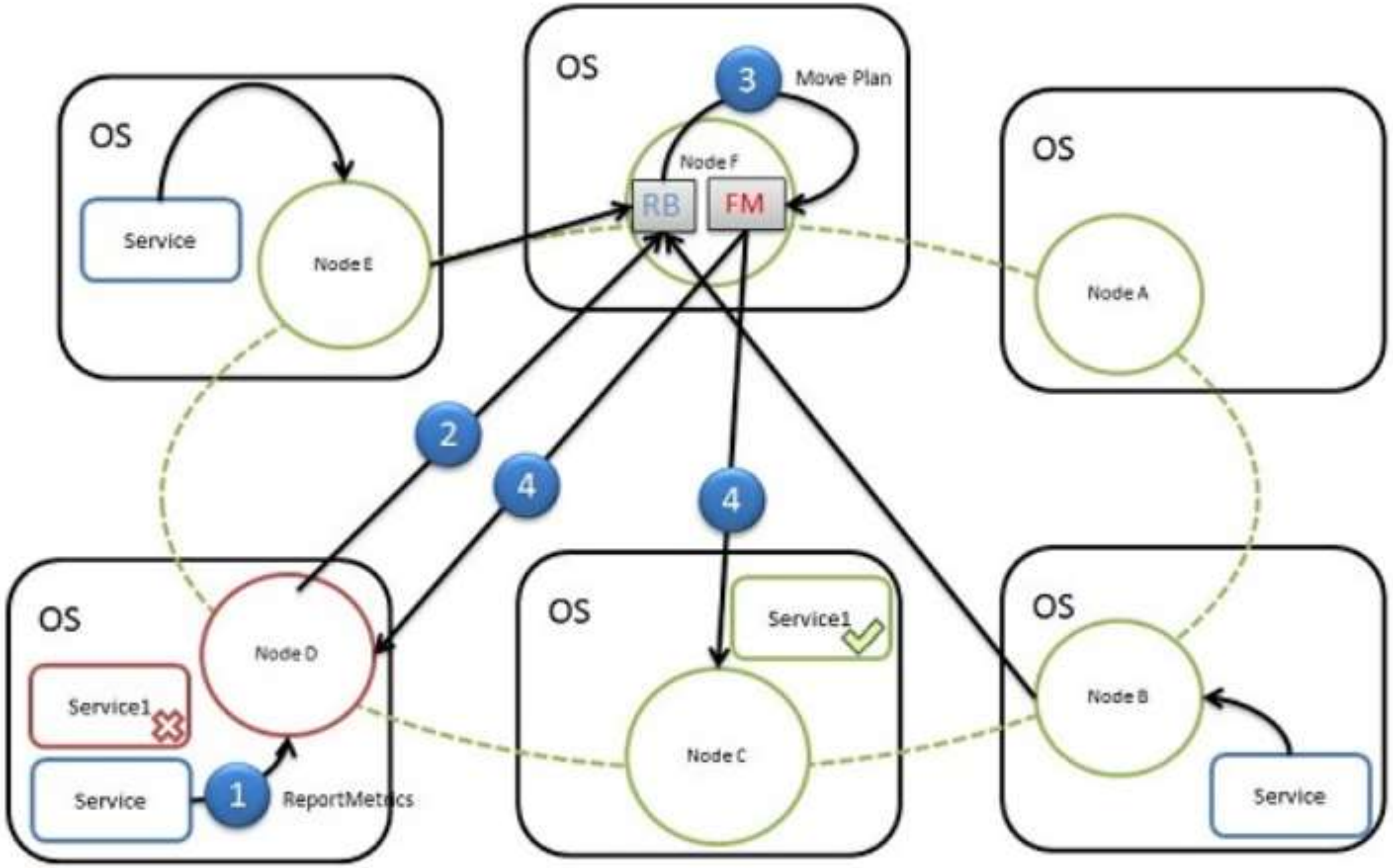
- Clusters are built on top of virtual machine scale sets, and auto-scaling is now available
- https://azure.microsoft.com/en-us/documentation/articles/service-fabric-cluster-scale-up-down/

Service Fabric – Internals

Resource Balancing

Microsoft Services

Resource Balancer architecture overview



Placement Constraints

- Placement Constraints
  - Used to indicate on which nodes certain services should run
  - Extensible by users – tag nodes with custom properties and use these properties for placement criteria
  - The constraint statement is at the service level
  - If a constraint applies to all nodes, apply via ARM or ClusterManifest.xml
- When a constraint is applied to the ServiceManifest.xml, you can use New/Update-ServiceFabricService to apply the constraint

  "(Constraint == Americas) && (FrontEnd == false)"

Node Placement properties

- Key/Value tags on nodes are known as ‘node placement properties’
- Value can be a string, bool or signed long
- Property Example:
    - Hardware: *Type of* CPU, RAM, disk, network, GPU, etc.
    - Other: Geolocation, network access/perimeter network
    - <Property Name="Continent" Value="Americas"/>
- Default properties include:
    - NodeType
    - NodeName
    - FaultDomain
    - UpdateDomain

Placement Constraint example

- In the ClusterManifest.xml file

```
<NodeType Name="NodeType01">
    <PlacementProperties>
      <Property Name="HasSSD" Value="true"/>
      <Property Name="NodeColor" Value="green"/>
      <Property Name="SomeProperty" Value="5"/>
    </PlacementProperties>
  </NodeType>
```

- In your code

```
FabricClient fabricClient = new FabricClient();
StatefulServiceDescription serviceDescription = new StatefulServiceDescription();
serviceDescription.PlacementConstraints = "(HasSSD == true && SomeProperty >= 4)";
// add other required servicedescription fields
//...
await fabricClient.ServiceManager.CreateServiceAsync(serviceDescription);
```

Node Capacities

- A resource is known as a metric. A resource example would be memory or disk space
- Capacity is a constraint the cluster uses to determine how much of a resource a node has
- Set resource limits (*size of* disk, RAM, etc.) on desired nodes via Azure ARM or ClusterManifest.xml
- You can set capacities to individual nodes, not all nodes have that have the same setting
- Example:

```
<NodeType Name="NodeType02">
   <Capacities>
     <Capacity Name="MemoryInMb" Value="2048"/>
     <Capacity Name="Disk" Value="10000"/>
   </Capacities>
 </NodeType>
```

Service Load Metrics

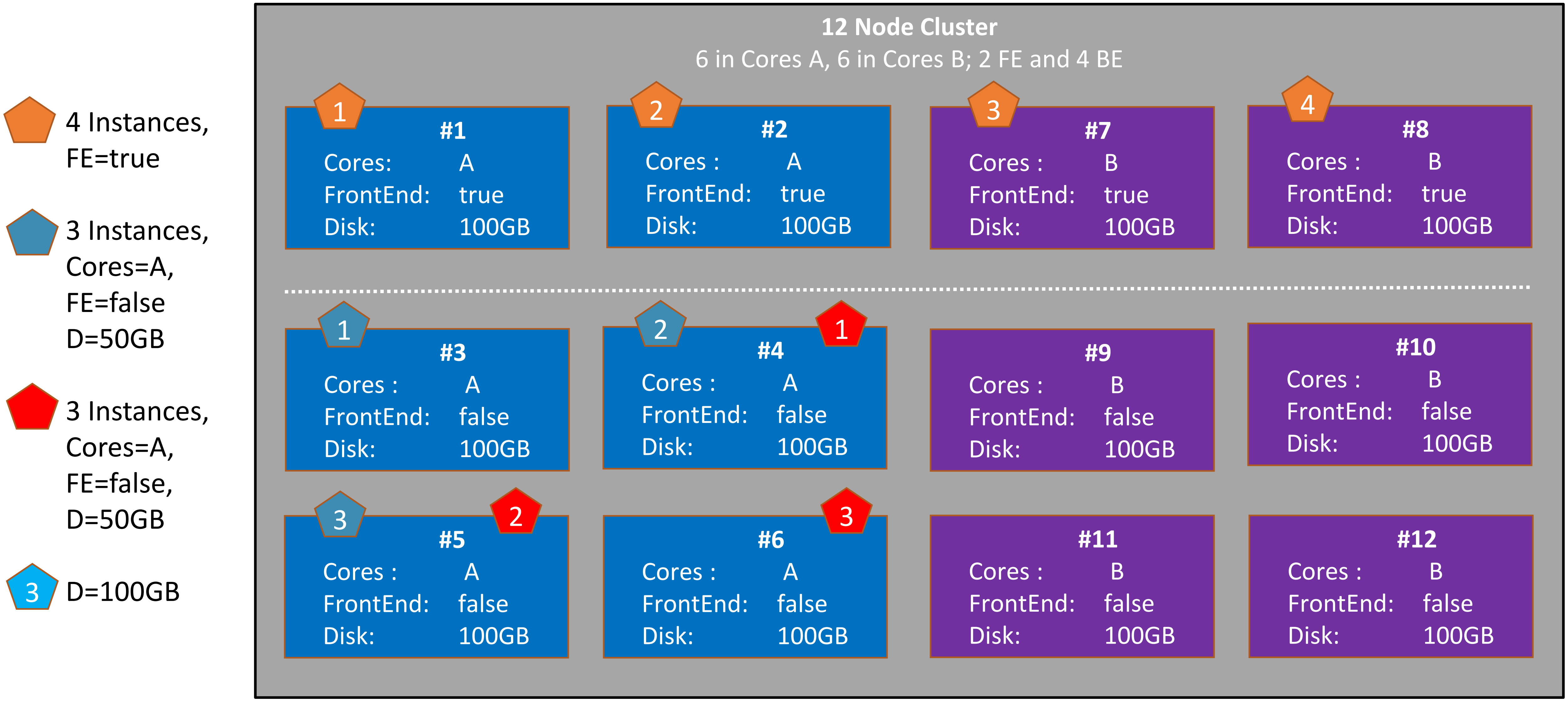
- Metric – anything you want to manage to deal with the performance of your services
- Example: Memory, Disk, CPU
- Default metrics: PrimaryCount, ReplicaCount and Count
- Specify metrics to balance and default load values via ServiceManifest.xml ~ stateful services

```
<LoadMetric Name="Disk" Weight="High" DefaultLoad="50"/>
```

- Override for a named service via New/Update-ServiceFabricService
  - Name, Weight (Importance: Zero, Low, Medium, High), Instances' value

    @("Disk,High,75", …)
  - SF prog models: code calls ReportLoad to update instance's values dynamically
- More depth - https://docs.microsoft.com/en-us/azure/service-fabric/service-fabric-cluster-resource-manager-metrics

Placement Resource Balancer



⬟ 4 Instances, FE=true

⬟ 3 Instances, Cores=A, FE=false D=50GB

⬟ 3 Instances, Cores=A, FE=false, D=50GB

⬟ 3 D=100GB

12 Node Cluster
6 in Cores A, 6 in Cores B; 2 FE and 4 BE

1 **#1**
Cores: A
FrontEnd: true
Disk: 100GB

2 **#2**
Cores : A
FrontEnd: true
Disk: 100GB

3 **#7**
Cores : B
FrontEnd: true
Disk: 100GB

4 **#8**
Cores : B
FrontEnd: true
Disk: 100GB

1 **#3**
Cores : A
FrontEnd: false
Disk: 100GB

2 1 **#4**
Cores : A
FrontEnd: false
Disk: 100GB

#9
Cores : B
FrontEnd: false
Disk: 100GB

#10
Cores : B
FrontEnd: false
Disk: 100GB

3 2 **#5**
Cores : A
FrontEnd: false
Disk: 100GB

3 **#6**
Cores : A
FrontEnd: false
Disk: 100GB

#11
Cores : B
FrontEnd: false
Disk: 100GB

#12
Cores : B
FrontEnd: false
Disk: 100GB

Node Movement

- Periodically, each node’s reconfiguration agent (RA) sends load values to the PRB* service
- PRB performs
  - Constraint check
    - If any constraint/capacity violated, moves instances to fix
    - This generally helps balance the cluster
  - Balance check
    - If cluster not balanced, moves instances (not being moved) to fix
- A service instance can report load against any metric but only specified metrics can be balanced against
  - Useful when upgrading code to report new metrics
  - Follow this up with an Update-ServiceFabricService

*PRB = Placement Resource Balancer

Simulated Annealing

- Service Fabric uses simulated annealing to improve the cluster's balance
- If cluster is imbalanced:
    - Give cluster's current balance a score
    - Generate a random, valid move and give it s score; keep best score
        - Repeat until some time period has elapsed
    - If final score is better than cluster's current score, initiate new balancing to incrementally improving the cluster's balance

Demonstration

Processes, Config and Stores

Demonstration

Service Fabric Failures

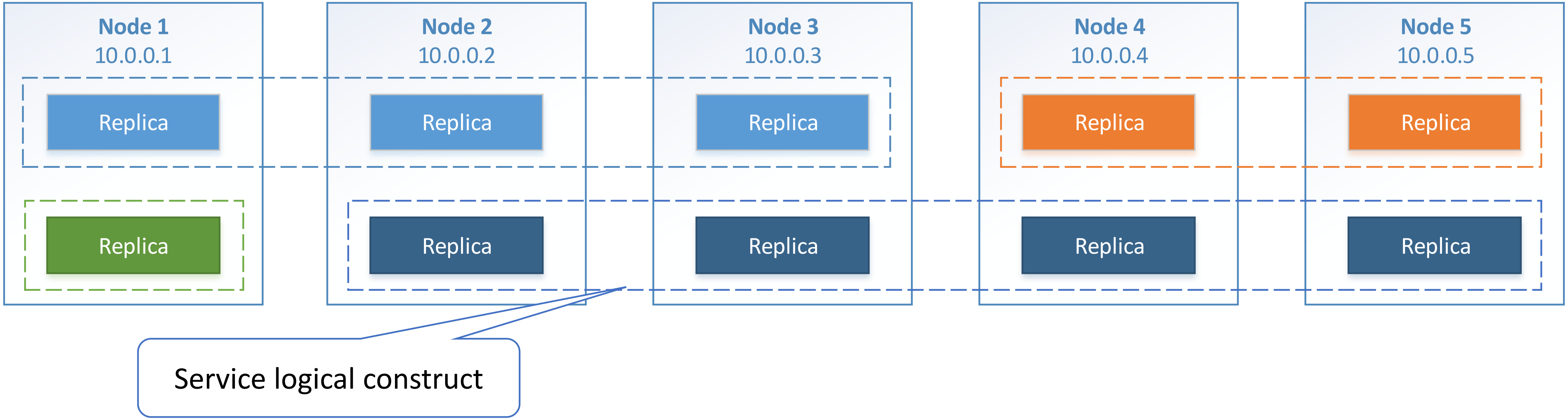
Service Fabric – Internals

Endpoints and Service Discovery

Microsoft Services

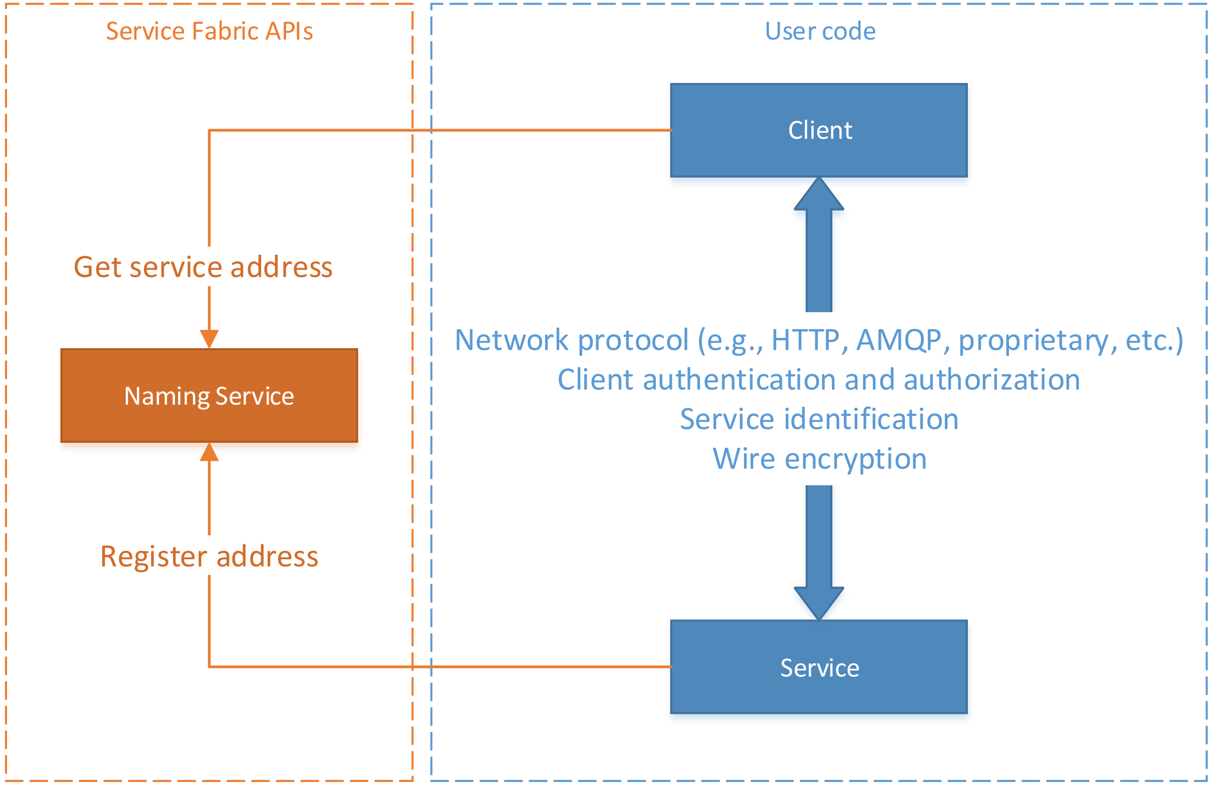
Service addressing

- A **service** is a logical construct that spans nodes. It does not have an address
- A **replica** of a service is placed on a node. It can have an address



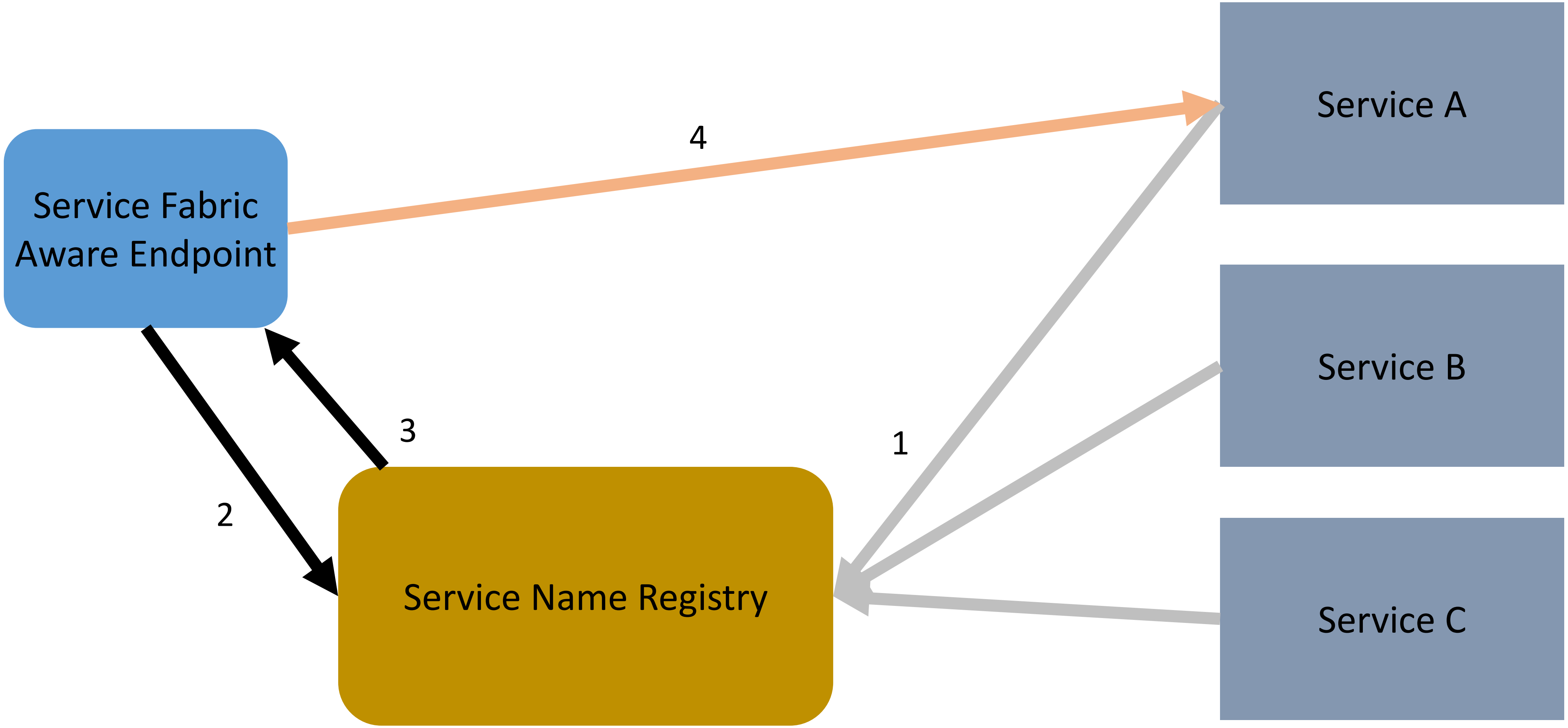
Service Communication

- Service Fabric only provides address resolution
- Clients and services are responsible for everything else



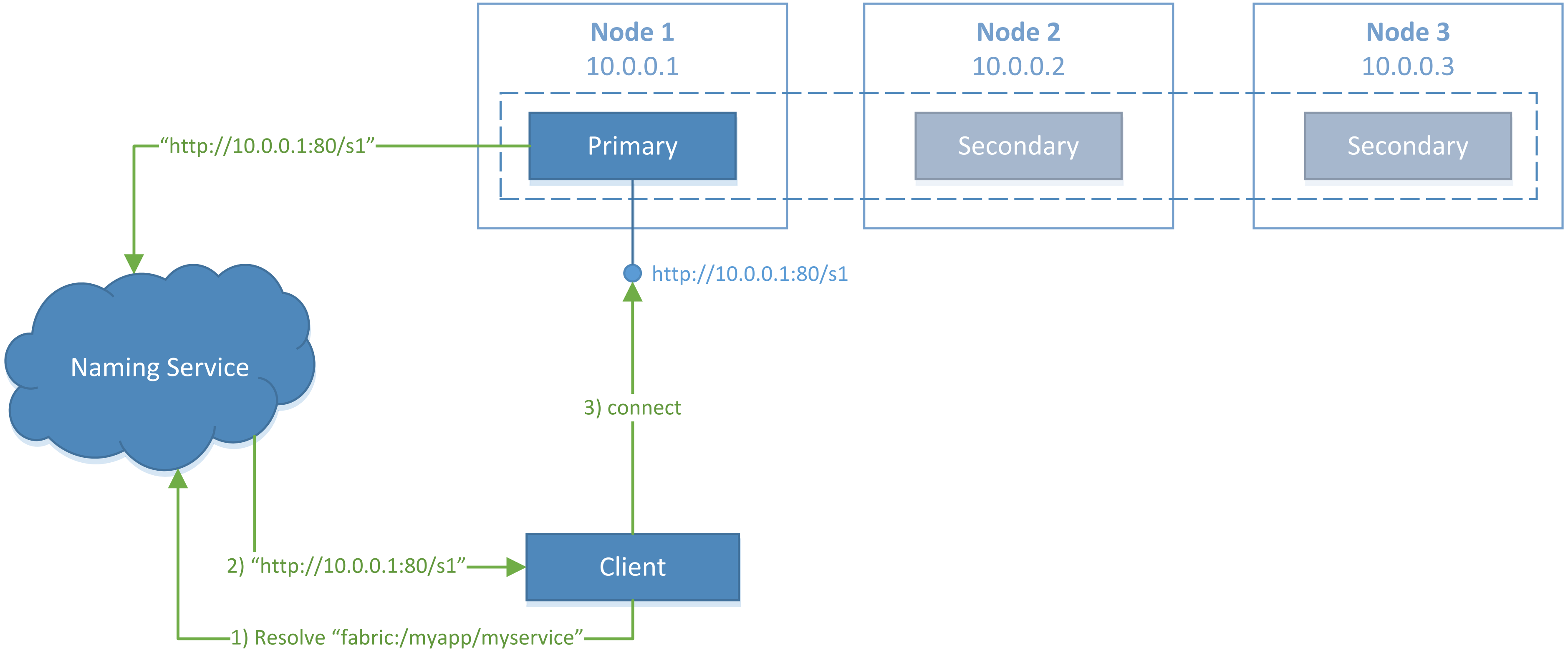
Network protocol (e.g., HTTP, AMQP, proprietary, etc.)
Client authentication and authorization
Service identification
Wire encryption

Service Discovery Pattern



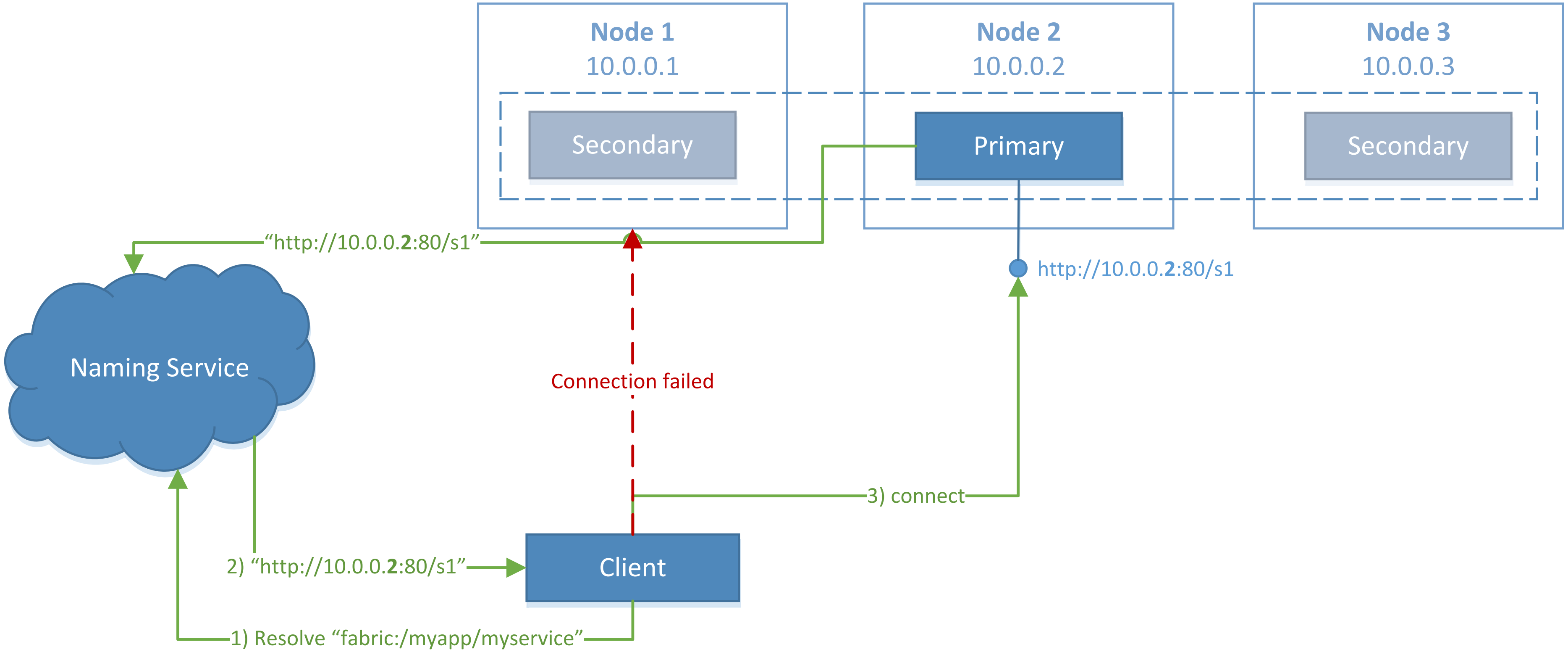
Stateful service communication

- A stateful service is not identical on all nodes
- Clients must find the correct replica to connect to



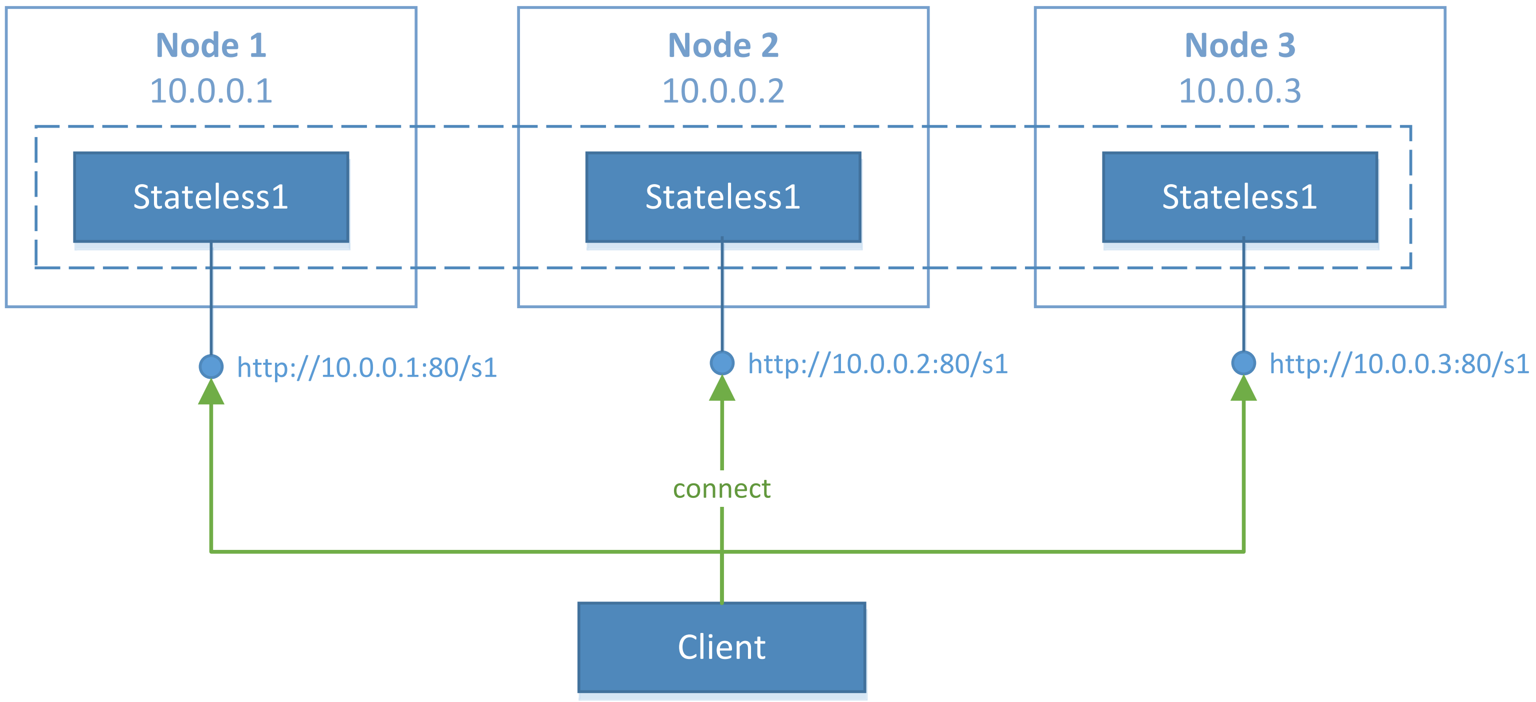
Stateful service communication

- Replicas can failover or move to a different node
- Clients need to re-resolve if a connection fails



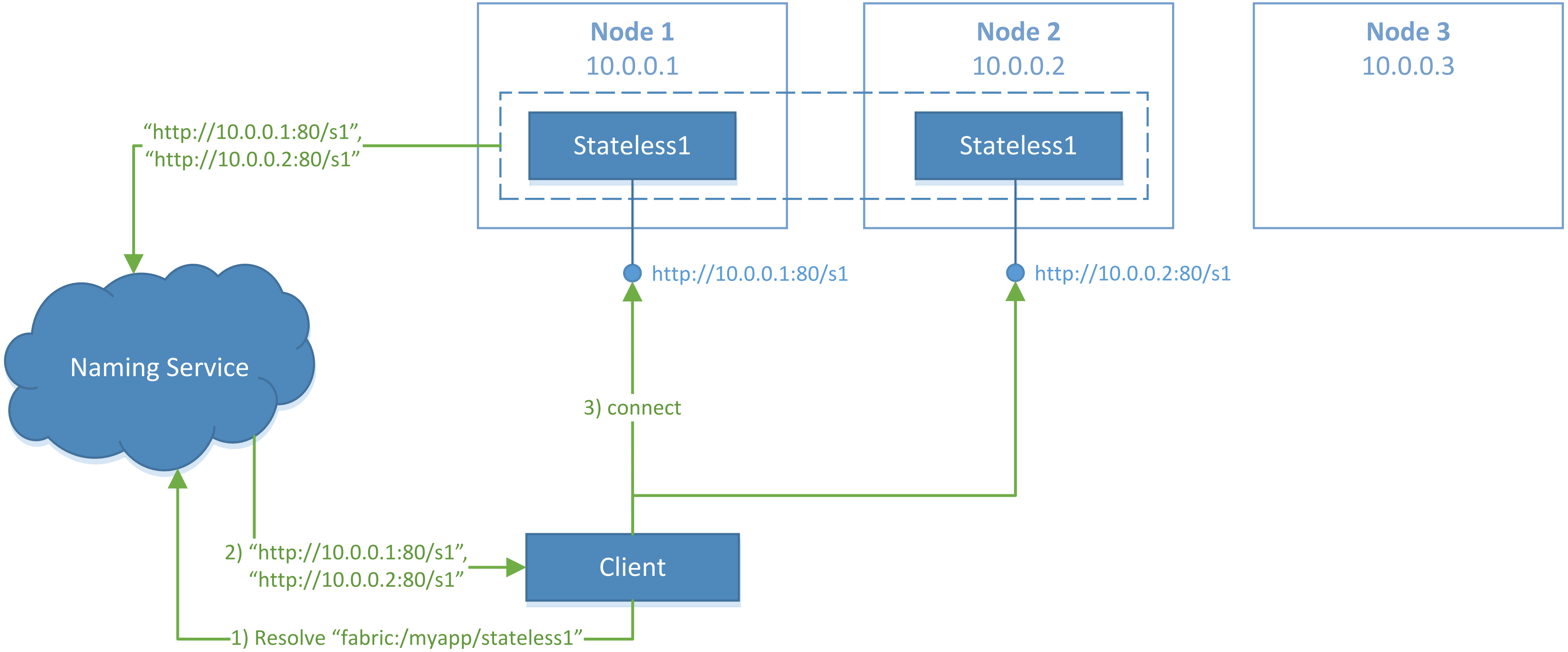
Stateless service communication

- A stateless service is identical on all nodes
- Clients can connect directly to any instance



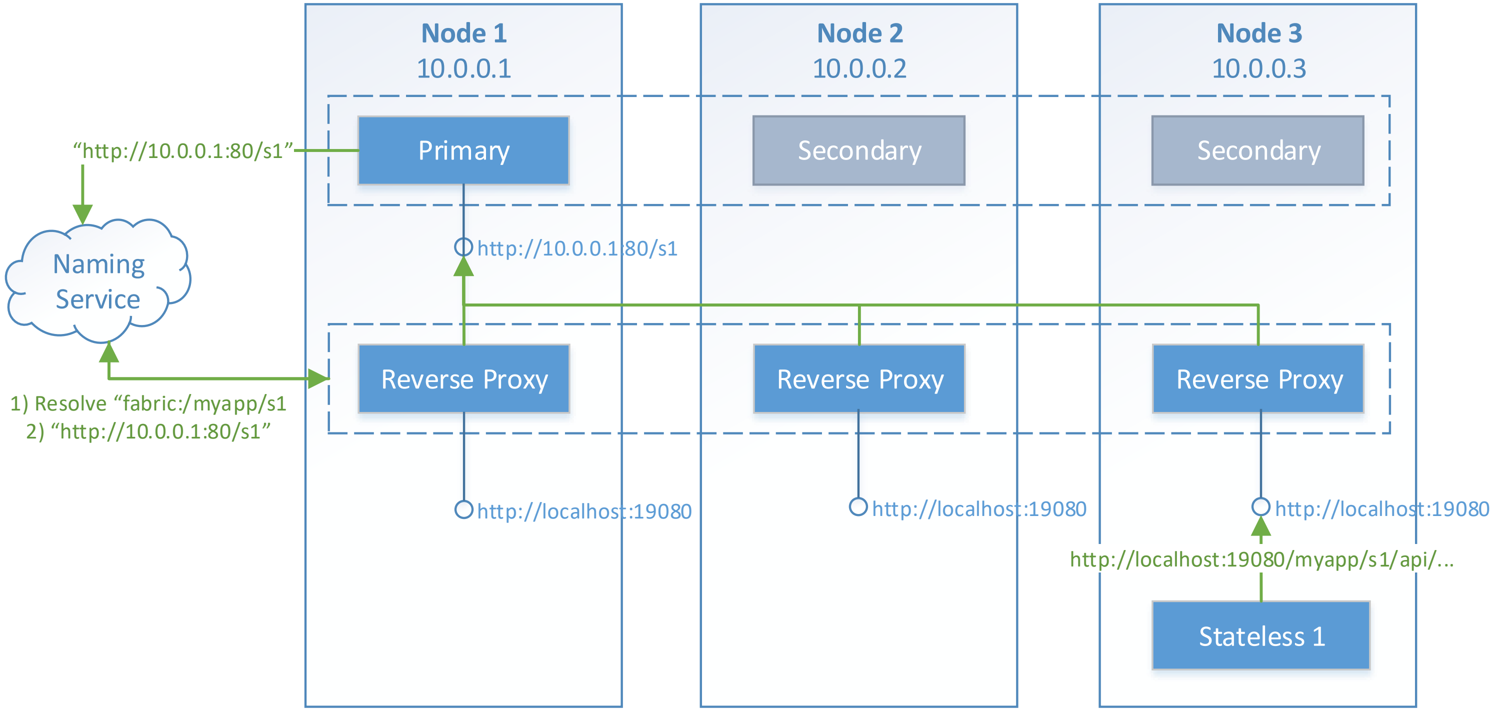
Stateless service communication

- Stateless services may not be running on all nodes
- Instances can still failover or move to a different node
- The client must find the instances' addresses



Service Fabric HTTP Reverse Proxy

- Don’t want to write complicated resolve code?
- Reverse Proxy does it for you and forwards requests
- The reverse proxy exists on each node as a service



Clients *must* retry failed network operations

- Retry a failed network operation when
  - Network fallacies (timeout, topology changes)
  - Server throttling
- *Don't* retry a failed network operation when
  - Service unavailable
  - Error reply
- Never assume a dependent service is already up and running
- To prevent clients from initiating a DDoS attack use
  - Exponential back-off & the circuit breaker pattern to prevent infinite retries

  https://msdn.microsoft.com/en-us/library/dn589784.aspx

Server *must* implement idempotent operations

- An operation is idempotent if it receives the same result when called multiple times
- Implementing idempotency is domain-specific:
    - Repeatedly creating a thumbnail of a specific photo produces the same result
    - Repeatedly subtracting $100 from a specific account produces different results
- HTTP requires most verbs be implemented idempotently

Service Instance Listeners

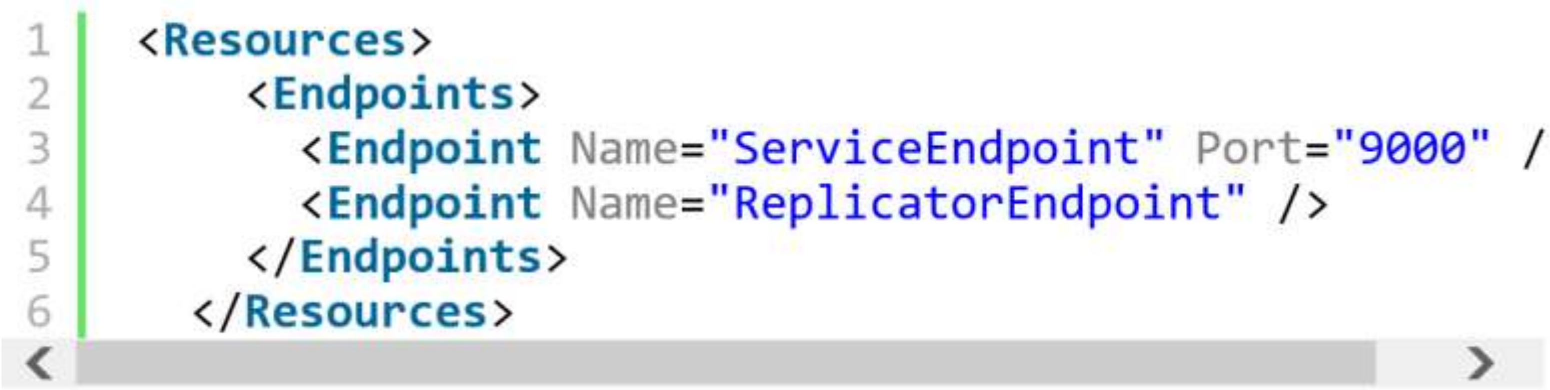
- When opened, each listener object
  - Listens on the node
  - Returns listener's name and opaque endpoint string to Service Fabric runtime
    - Service Fabric runtime sends all names/endpoints to Service Fabric naming service
- Clients query JSON with named service's partition's endpoints via
  - ResolvePartition (REST)
  - Resolve-ServiceFabricService (PowerShell)
  - ServicePartitionResolver (.NET) [called internally by service proxy]
  - Also visible in Service Fabric Explorer

Defining Endpoints

- Service returns a string from OpenAsync() method specifying the listening endpoint
- This string will be returned from the discovery service when requested by a client (for example using ServicePartitionResolver)
- If no port is specified for an internal service, service fabric will choose from a range of ports configured in the cluster manifest
- ClusterManifest.xml (section NodeTypes/NodeType/ApplicationEndPoints) ensures that the same port is not used on the same cluster node twice

Named Endpoints

Named end points forces your “ServiceEndPoint” to always use defined port (9000 in this case)
This port doesn’t have to reside within the range in the ClusterManifest.xml.



```
<Resources>
    <Endpoints>
      <Endpoint Name="ServiceEndpoint" Port="9000" /
      <Endpoint Name="ReplicatorEndpoint" />
    </Endpoints>
  </Resources>
```

Read requests on secondaries

- Secondaries make up a large part of the cluster and can add to processing power for reads
- Listening address returned from OpenAsync should include an extra guid to differentiate secondaries
- Requirements for clients
  - You can differentiate incoming requests based on (read only R or R/W) operations
  - You can route the requests to the correct replica based on the type (R vs. R/W)
  - You perform read only in your secondaries. Any write operation on your secondaries will result in an exception
- Enable by overriding *EnableCommunicationListenerOnSecondary* to return true

Demonstration

Service Communication

Microsoft