prodot GmbH

Service Fabric - Reliable Services

21.09.2017

Agenda

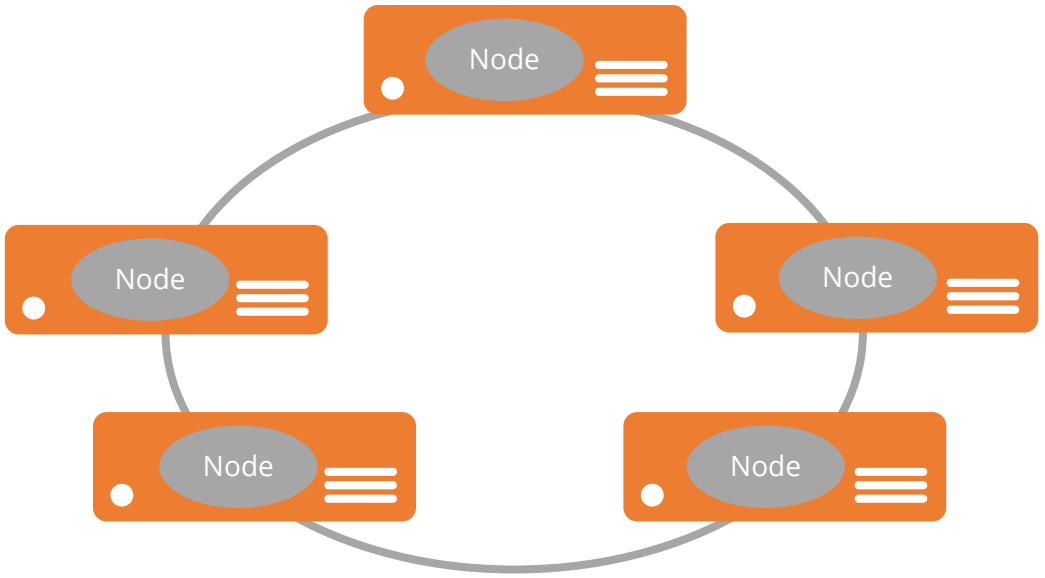
- Service Fabric - Rückblick
- Reliable Services
- Statefull oder Stateless?
- Kommunikation
- Verlässlichkeit
- Skalierung

Service Fabric - Rückblick

Service Fabric

Rückblick

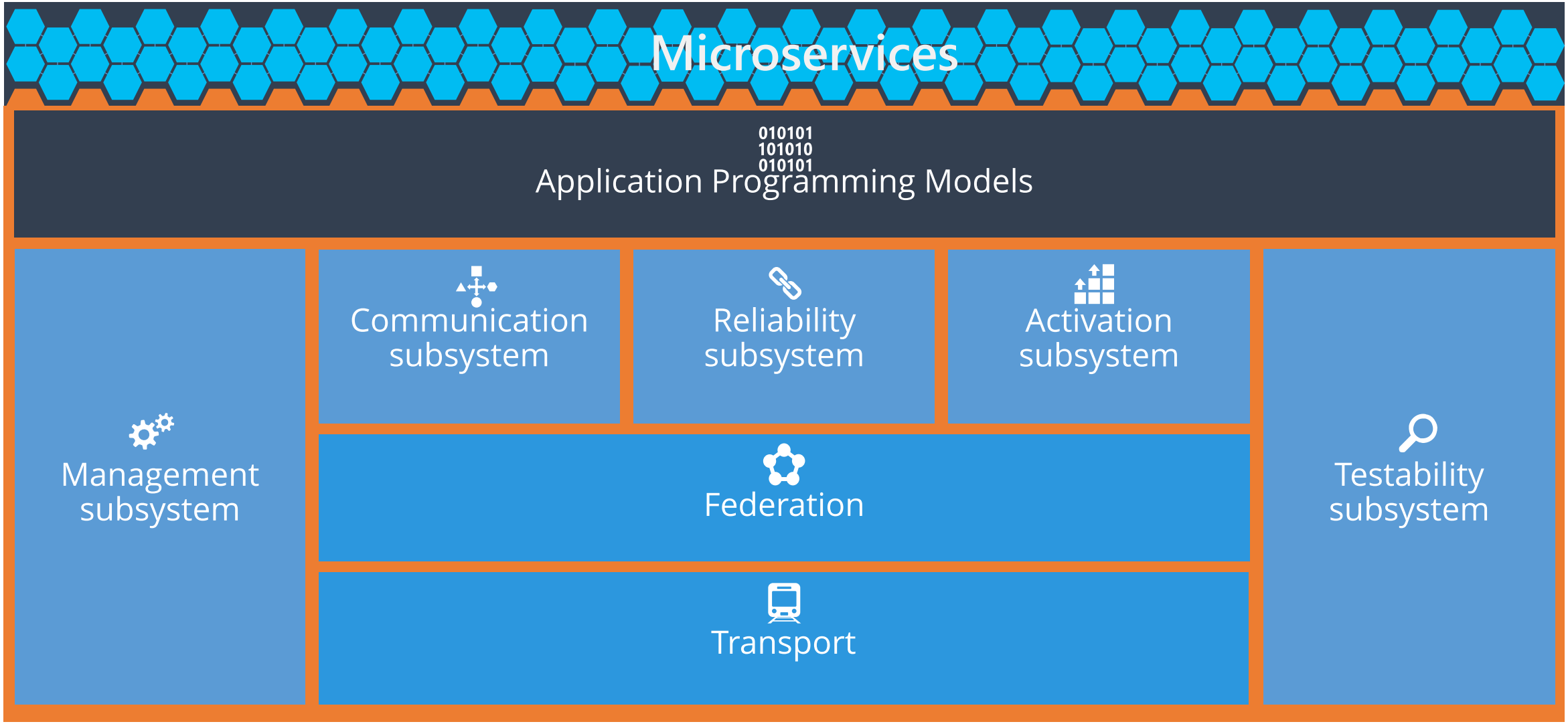
- Verteilte Microservice-Plattform
- Verteilt / Orchestriert / Skaliert Microservices
- Hosted Applikationen
- Hochverfügbar und ausfallsicher



Service Fabric

Rückblick

- Drei bis zu Tausende OS-Instanzen bilden einen Cluster
- Maschinen können virtuell oder physikalisch sein
- Cloud und Betriebssystem agnostisch*
- Naming- , Monitoring- , Lifecycle-Management-Dienste



Service Fabric - Komponenten

Reliable Services

Verwenden ein minimales Framework zur Entwicklung von Services.

Die bereitgestellten APIs ermöglichen die Interaktion mit anderen Services und dem Cluster.

(C# / Java)

Actors

Bauen auf Reliable Services auf, sind aber auf Single Threads und eine spezifische Funktion beschränkt. Ermöglichen, je nach Anwendungszweck eine effizientere Skalierung.

(C# / Java)

Guest executables

Laufen als externe Prozesse und können nur mittels der bereitgestellten REST-APIs mit dem Cluster interagieren.

Sind unabhängig von den verwendeten Programmiersprachen und Frameworks.

Reliable Services

Reliable Services

Vorteile gegenüber normalen Diensten

- Zuverlässig
- Verfügbar
- Skalierbar
- Konsistent

Reliable Services

Features

- Wahl zwischen Stateful- und Stateless-Services
- Zugriff auf die Service Fabric APIs
- Kommunikation über bel. Transport-Schichten und Protokolle
- Verwalten des States über Reliable Collections

Stateful oder Stateless

Reliable Services

Stateful oder Stateless?

Stateless

- Einfache Funktionalität
- Unabhängig und wiederholbar
- Einfacher zu Entwickeln
- Beispiel: WebAPIs / SPAs

Stateful

- Komplexere Funktionen
- Höhere Laufzeit
- State muss dauerhaft gespeichert werden und schnell abrufbar sein
- Beispiel: Videokonvertierung, Raytracing, Wettersimulationen

Reliable Services

Stateless

```
...public abstract class StatelessService : IStatelessUserServiceInstance
{
    ...protected StatelessService(StatelessServiceContext serviceContext);

    ...public StatelessServiceContext Context { get; }
    ...protected IStatelessServicePartition Partition { get; }

    ...protected virtual IEnumerable<ServiceInstanceListener> CreateServiceInstanceListeners();
    ...protected IReadOnlyDictionary<string, string> GetAddresses();
    ...protected virtual void OnAbort();
    ...protected virtual Task OnCloseAsync(CancellationToken cancellationToken);
    ...protected virtual Task OnOpenAsync(CancellationToken cancellationToken);
    ...protected virtual Task RunAsync(CancellationToken cancellationToken);
}
```

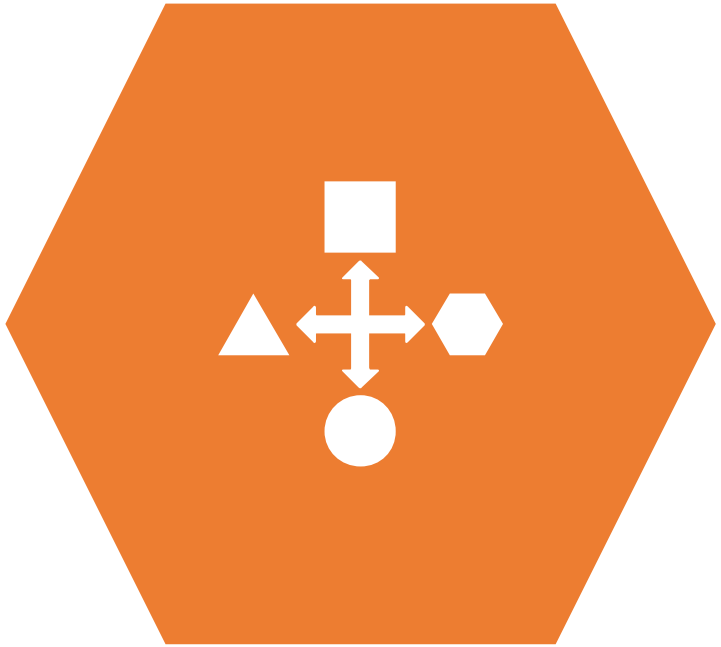
Reliable Services

Stateful

```
public abstract class StatefulService : StatefulServiceBase
{
    protected StatefulService(StatefulServiceContext serviceContext);
    protected StatefulService(StatefulServiceContext serviceContext, IReliableStateManagerReplica reliableStateManagerReplica);

    public IReliableStateManager StateManager { get; }
}
```

```
public abstract class StatefulServiceBase : IStatefulUserServiceReplica
{
    protected StatefulServiceBase(StatefulServiceContext serviceContext, IStateProviderReplica stateProviderReplica);

    public StatefulServiceContext Context { get; }
    protected IStatefulServicePartition Partition { get; }

    public Task BackupAsync(BackupDescription backupDescription);
    public Task BackupAsync(BackupDescription backupDescription, TimeSpan timeout, CancellationToken cancellationToken);
    protected virtual IEnumerable<ServiceReplicaListener> CreateServiceReplicaListeners();
    protected IReadOnlyDictionary<string, string> GetAddresses();
    protected virtual void OnAbort();
    protected virtual Task OnChangeRoleAsync(ReplicaRole newRole, CancellationToken cancellationToken);
    protected virtual Task OnCloseAsync(CancellationToken cancellationToken);
    protected virtual Task<bool> OnDataLossAsync(RestoreContext restoreCtx, CancellationToken cancellationToken);
    protected virtual Task OnOpenAsync(ReplicaOpenMode openMode, CancellationToken cancellationToken);
    protected virtual Task RunAsync(CancellationToken cancellationToken);
}
```

Kommunikation



Reliable Services

Kommunikation - Bring your own protocol

- Bel. Transportschicht z.B. TCP / UDP
- Bel. Protokolle z.B. HTTP / HTTPS / FTP
- Bel. Formate z.B. JSON / XML / Protocol Buffers

Reliable Services

Kommunikation - Listeners

```
...public sealed class ServiceInstanceListener
{
    ...public const string DefaultName = "";

    ...public ServiceInstanceListener(Func<StatelessServiceContext, ICommunicationListener> createCommunicationListener,
        string name = "");

    ...public string Name { get; }
    ...public Func<StatelessServiceContext, ICommunicationListener> CreateCommunicationListener { get; }
}
```

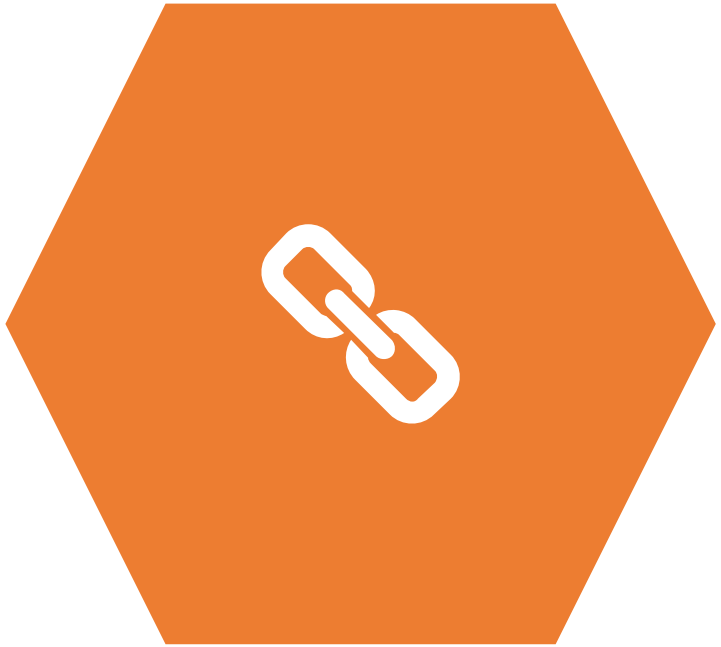
Reliable Services

Kommunikation - Listeners

```
public interface ICommunicationListener
{
    void Abort();
    Task CloseAsync(CancellationToken cancellationToken);
    Task<string> OpenAsync(CancellationToken cancellationToken);
}
```

```
Task<string> ICommunicationListener.OpenAsync(CancellationToken cancellationToken)
{
    var endpoint = FabricRuntime.GetActivationContext().GetEndpoint(_endpointName);
    var serverUrl = $"{endpoint.Protocol}://+:{endpoint.Port}";

    _webHost = CreateWebHost(endpoint.Protocol, endpoint.Port);
    _webHost.Start();

    return Task.FromResult(serverUrl);
}
```

Verlässlichkeit



Reliable Services

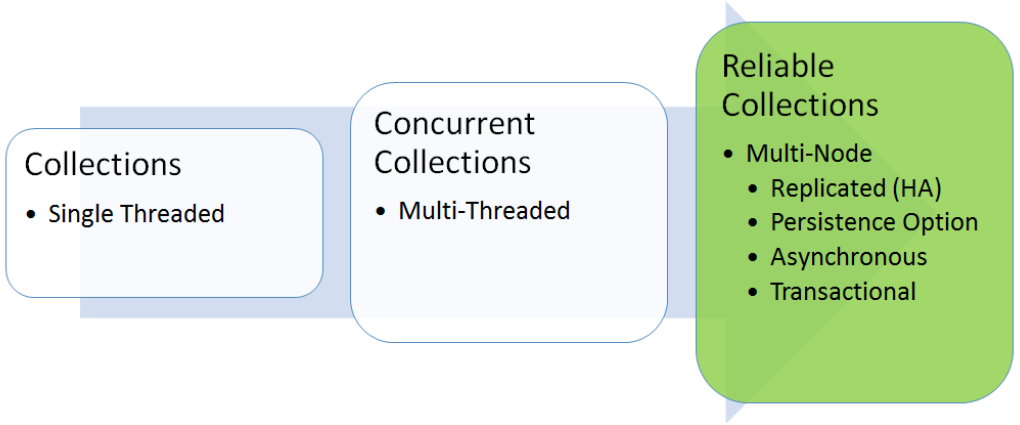
Stateful Services – Verlässlichkeit / State

Der State eines Stateful Service kann verwaltet werden über:

- StateManager
- Reliable Collections

Dabei sind:

- Lesezugriffe immer lokal
- Schreibzugriffe über den Cluster verteilt



Reliable Services

Stateful Services – Reliable Collections

- Zusätzlich bietet Service Fabric verteilte Collection-Typen:
  - Reliable Dictionary
  - Reliable Queue
  - Reliable Concurrent Queue
- Sie stellen ohne größeren Aufwand Daten über alle Nodes bereit und sind dabei:
  - Repliziert
  - Persistiert
  - Asynchron
  - Transaktionssicher

Skalierung

Reliable Services

Skalierung – Kann stattfinden über

- Anzahl der StatelessService-Instanzen
- Hinzufügen/Entfernen von bestimmten Instanzen
- Hinzufügen/Entfernen von Applikationen
- Partitionierung der Services nach Eingangsdaten
- Anzahl der Nodes im Cluster
- Über Azure Virtual Machine Scale Sets (VMSS)

Reliable Services

Skalierung – Partitionierung nach

- Intervall (Uniform Ranged – Int64-Indikator)
- Namen
- Singleton

Demo

Service Fabric

Getting Started – Demo / Repository

https://github.com/Azure-Samples/service-fabric-dotnet-getting-started/

Thank you.

People. Passion. Performance.

prodot®

Software for
Market Leaders

prodot GmbH
Schifferstraße 196
47059 Duisburg
Germany

Telephone +49 (203) 34 69 45 – 0
Fax +49 (203) 34 69 45 - 20

info@prodot.de
https://prodot.de