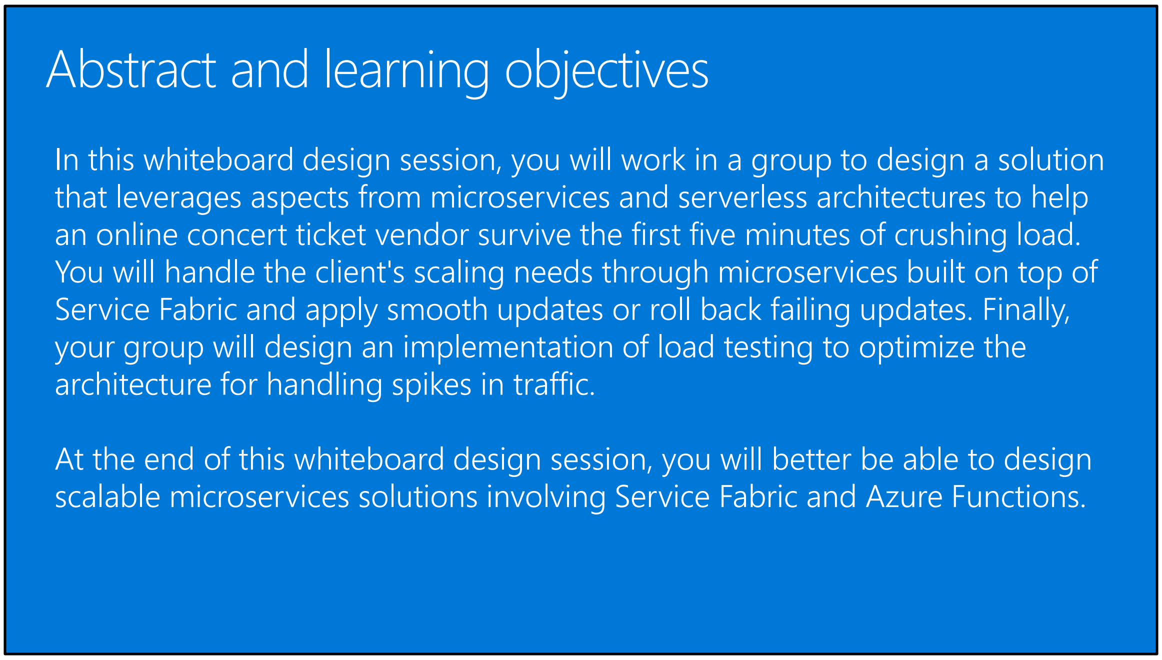
Microsoft

Microsoft Cloud Workshop

Microservices architecture

Alex Mang

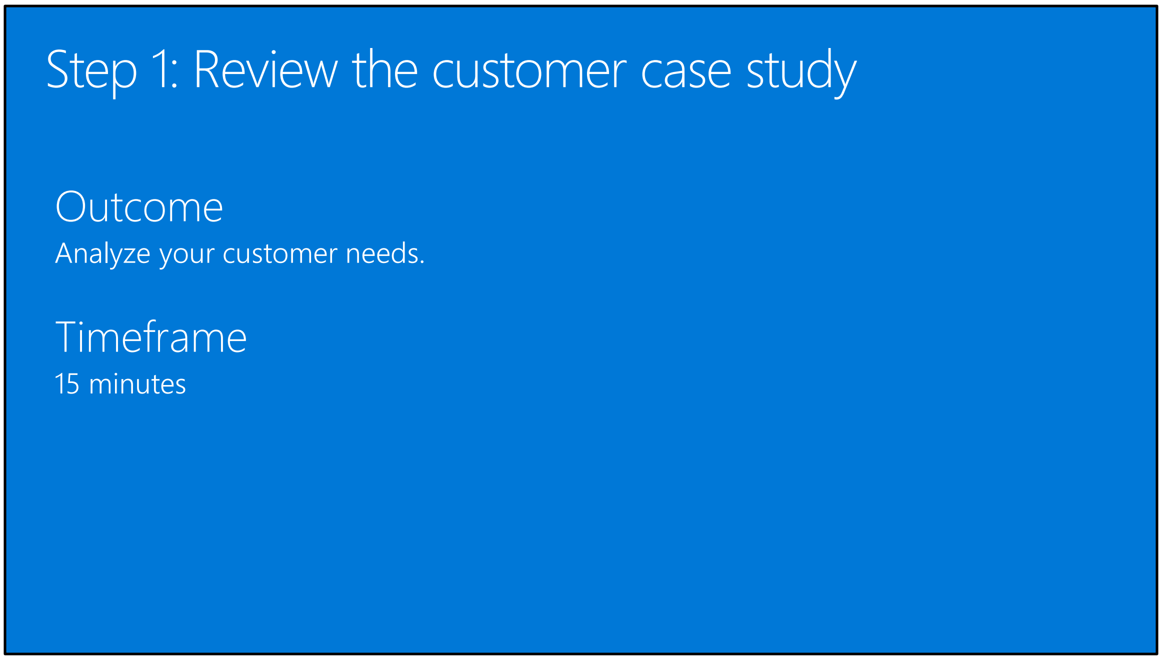


Abstract and learning objectives

In this whiteboard design session, you will work in a group to design a solution that leverages aspects from microservices and serverless architectures to help an online concert ticket vendor survive the first five minutes of crushing load. You will handle the client's scaling needs through microservices built on top of Service Fabric and apply smooth updates or roll back failing updates. Finally, your group will design an implementation of load testing to optimize the architecture for handling spikes in traffic.

At the end of this whiteboard design session, you will better be able to design scalable microservices solutions involving Service Fabric and Azure Functions.

This is an introduction to the Cloud Workshop at a high level. Later we'll get into customer objections, requirements, etc… but we want to ground the participants on the business outcomes we're going after for the day.



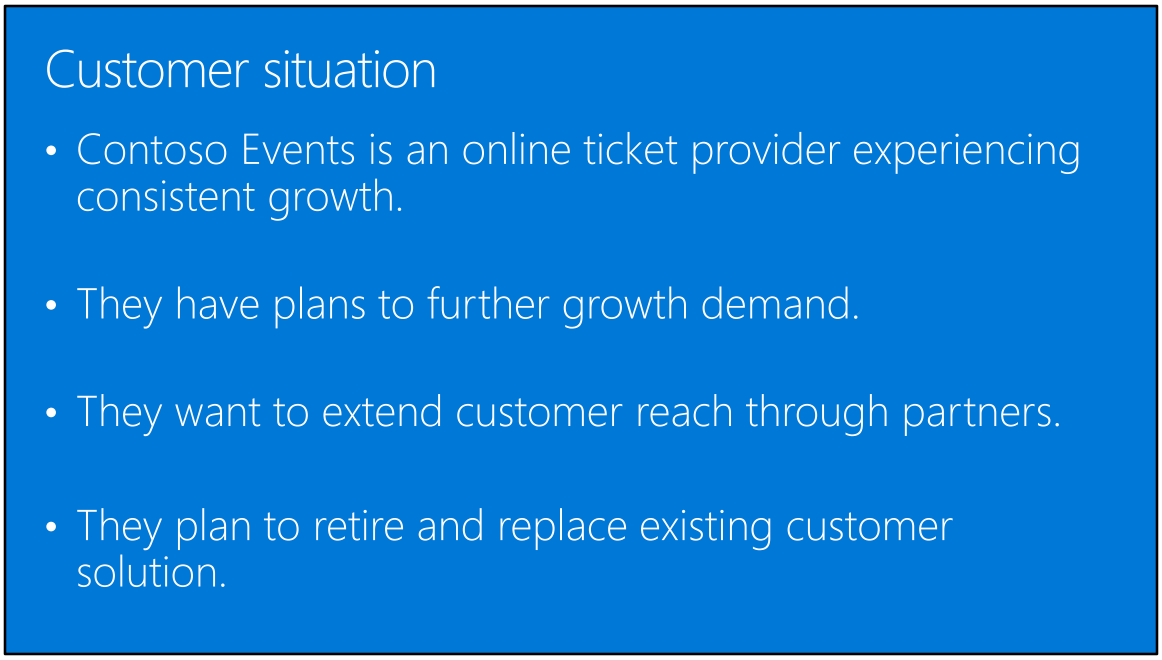
Step 1: Review the customer case study

Outcome

Analyze your customer needs.

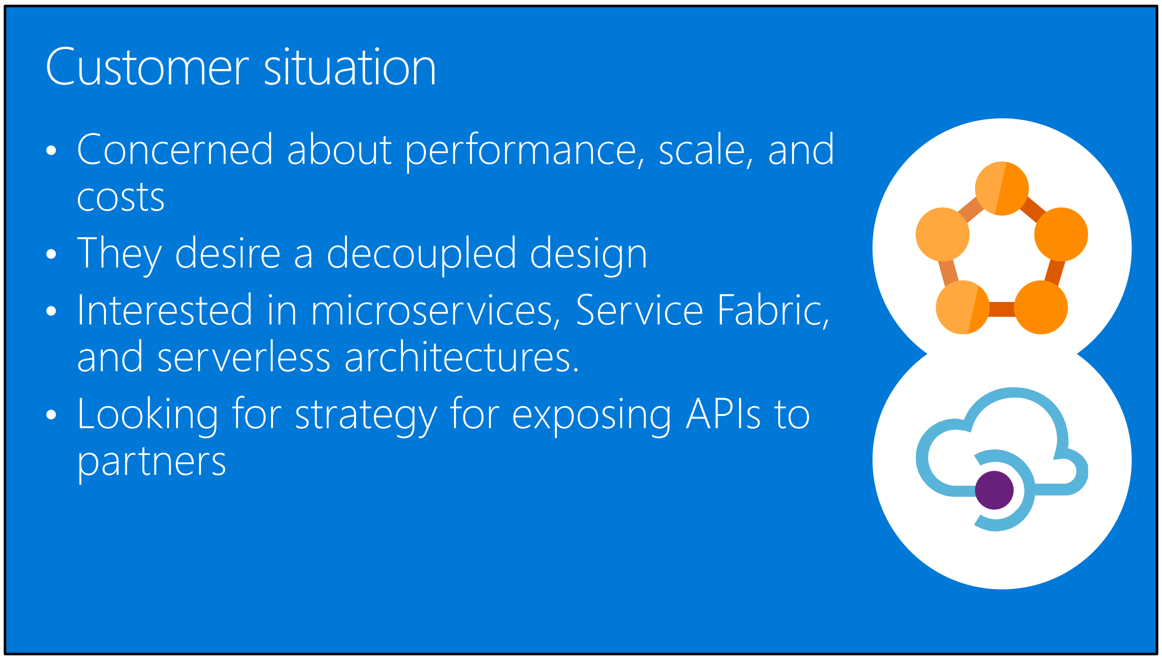
Timeframe

15 minutes



Customer situation

- Contoso Events is an online ticket provider experiencing consistent growth.
- They have plans to further growth demand.
- They want to extend customer reach through partners.
- They plan to retire and replace existing customer solution.



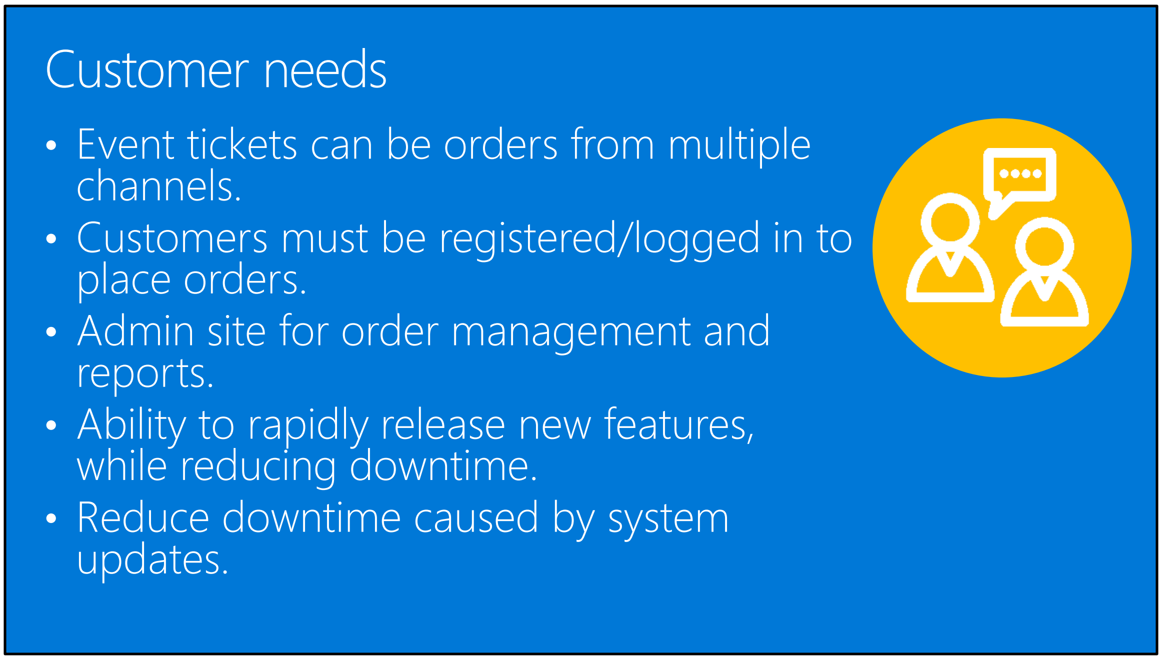
Customer situation

- Concerned about performance, scale, and costs
- They desire a decoupled design
- Interested in microservices, Service Fabric, and serverless architectures.
- Looking for strategy for exposing APIs to partners

them to schedule down time to ensure safe deployment.

- The CIO has heard about microservices, and is interested in exploring how Service Fabric, Azure Functions and supporting Azure features may help the team:
    - Increase agility in their development cycle
    - Decrease impact across features
    - Support continuous delivery
    - Keep operations lean
    - Keep costs contained during peak loads

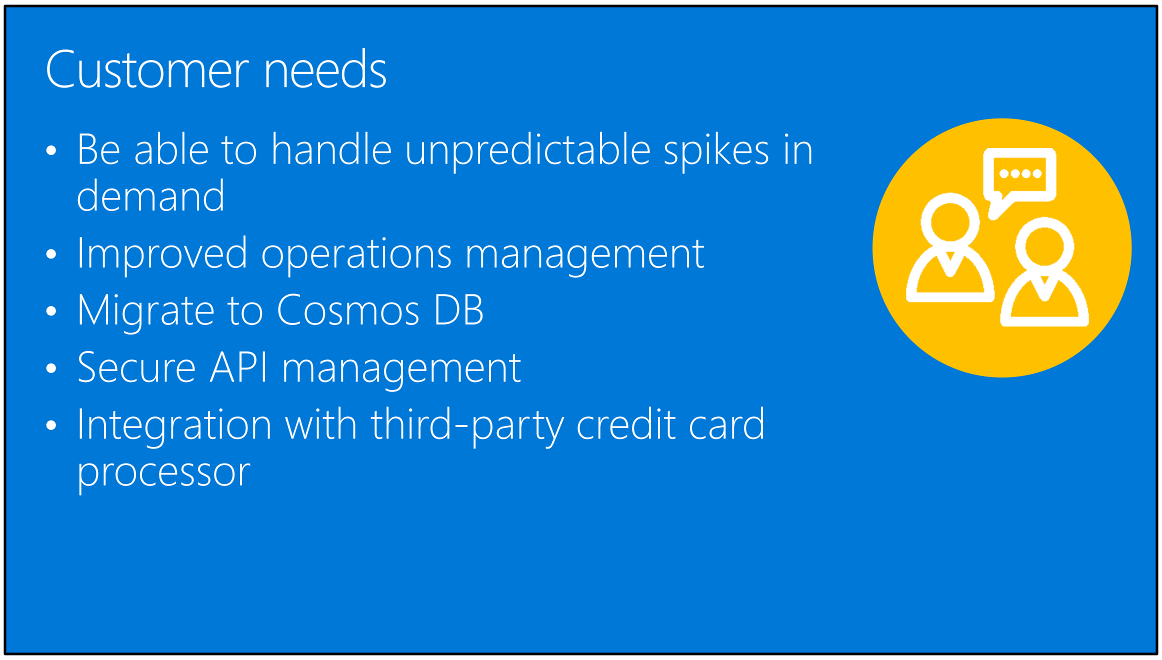
- In addition, they want a solid strategy for exposing APIs to partners in a secure and manageable way



Customer needs

- Event tickets can be orders from multiple channels.
- Customers must be registered/logged in to place orders.
- Admin site for order management and reports.
- Ability to rapidly release new features, while reducing downtime.
- Reduce downtime caused by system updates.

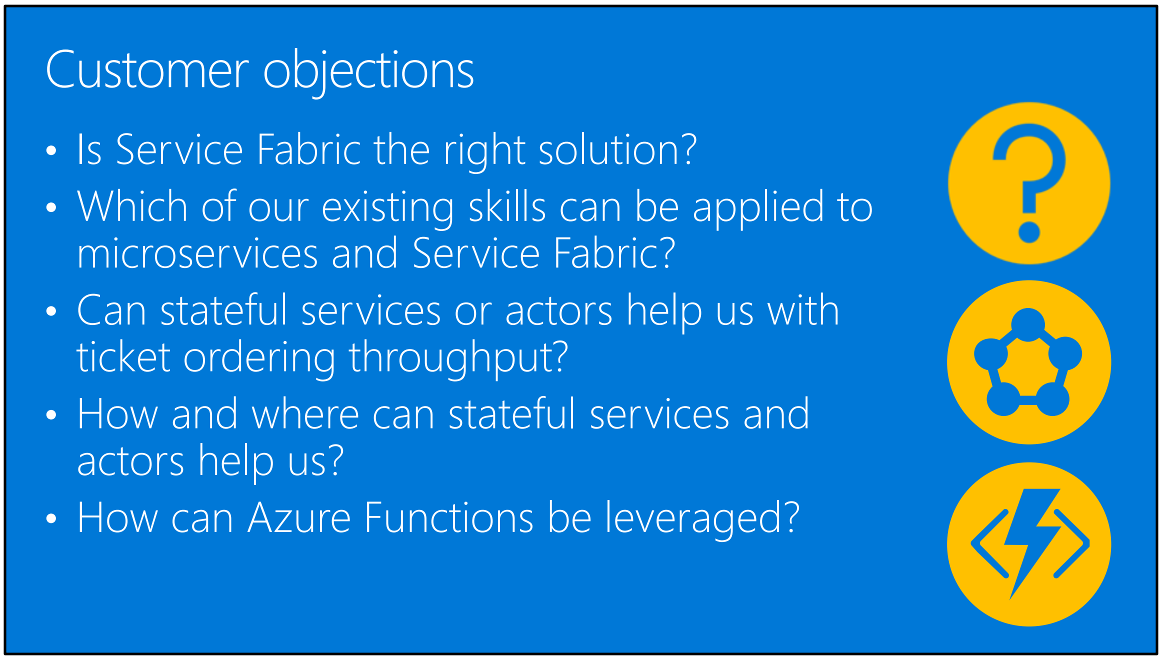
- Event tickets can be ordered from multiple channels: the web site, new mobile applications, and third-party site and applications via available APIs.
- Customers must be registered / logged in to place orders, so that they can login and find their orders, and for reporting and analytics purposes.
- Internal staff will manage orders and view reports from the Admin site.
- The ability to rapidly release new features that may involve UI, business logic and data model changes. Reduce dependency across features.
- Reduced overall downtime caused by system updates. Rollouts must be possible without scheduled downtime. Rollbacks must be possible in the event of failure.



Customer needs

- Be able to handle unpredictable spikes in demand
- Improved operations management
- Migrate to Cosmos DB
- Secure API management
- Integration with third-party credit card processor

- The solution must be able to handle increased system load for ticket purchasing including higher peak periods without excessive increases in management overhead and cost.
- Operations management overhead must be improved through better system monitoring, visibility, self-healing services and auto-scale features.
- The customer has decided to migrate from SQL Server to Cosmos DB for a more flexible schema and increased scalability across features.
- A solution for securing and managing APIs used internally and by external partners with ability to easily publish, version, onboard consumers, control policy, monitor and audit usage.
- The solution currently processes credit cards with a third-party payment-processing provider. This aspect of the solution will remain the same but require integration into the new design.



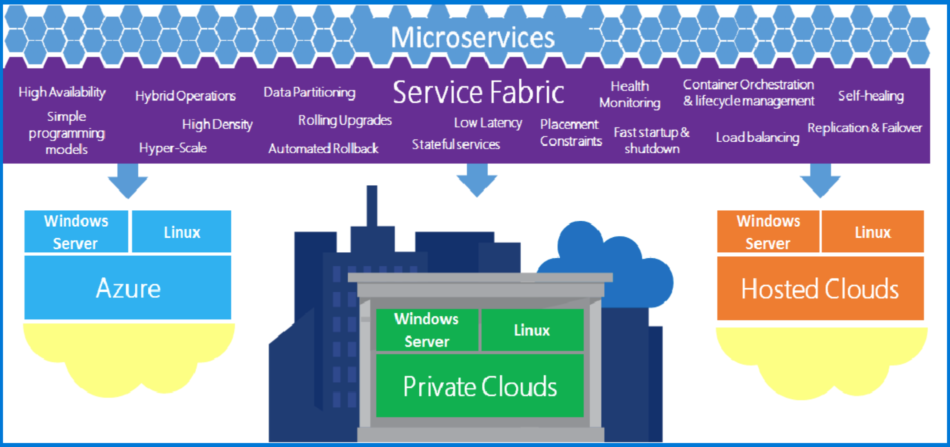
Customer objections

- Is Service Fabric the right solution?
- Which of our existing skills can be applied to microservices and Service Fabric?
- Can stateful services or actors help us with ticket ordering throughput?
- How and where can stateful services and actors help us?
- How can Azure Functions be leveraged?

- While we are interested in the microservices approach, we are still comparing Service Fabric with PaaS features such as App Services and SQL DB. Service Fabric seems relatively new, while App Services and SQL DB have been around for some time.
- Microservices architectures are completely new to the Contoso Events team. If we were to go forward with Service Fabric, we would like to understand what skills the team can carry forward, and how much of a learning curve exists.
- We'd like to understand if stateful services or stateful actors will help us with ticket ordering throughput, workflow and state management, and easier rollouts of changes to this process.
- We are not clear how and where to incorporate stateful services and actors alongside other storage such as Cosmos DB. We need the ability to support robust ad-hoc queries against our system data such as events, customers, orders and related metrics – but would like to take advantage of the performance and reliability of Service Fabric stateful options as well.
- Could we consider Azure Functions as an alternative back end implementation for our APIs?

Common scenarios

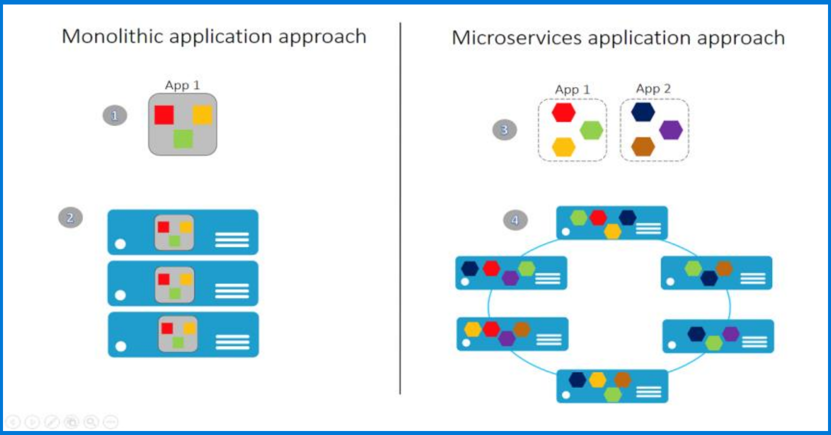
Service Fabric overview



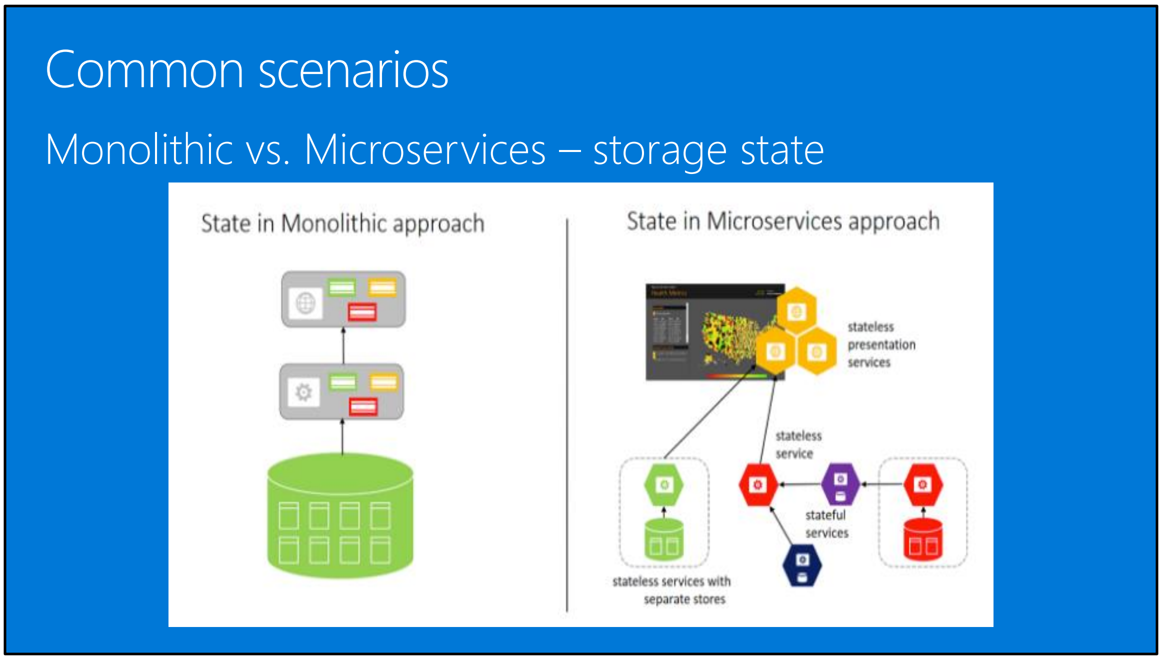
This diagram represents a Service Fabric overview for this type of scenario, from which you can draw inspiration. You will find this diagram within the Whiteboard Design Session Student Guide.

Common scenarios

Monolithic vs. Microservices approach

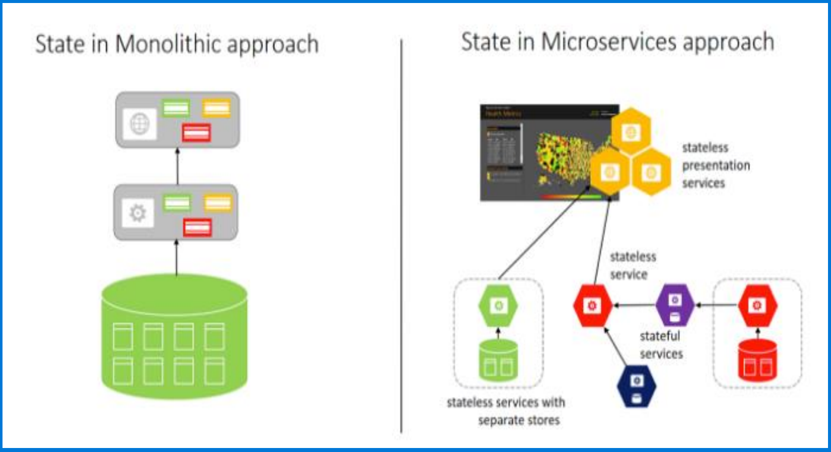

Monolithic application approach
App 1
1
2
Microservices application approach
App 1
App 2
3
4

This diagram presents a comparison of monolithic versus microservices approaches for this type of scenario, from which you can draw inspiration. You will find this diagram within the Whiteboard Design Session Student Guide.

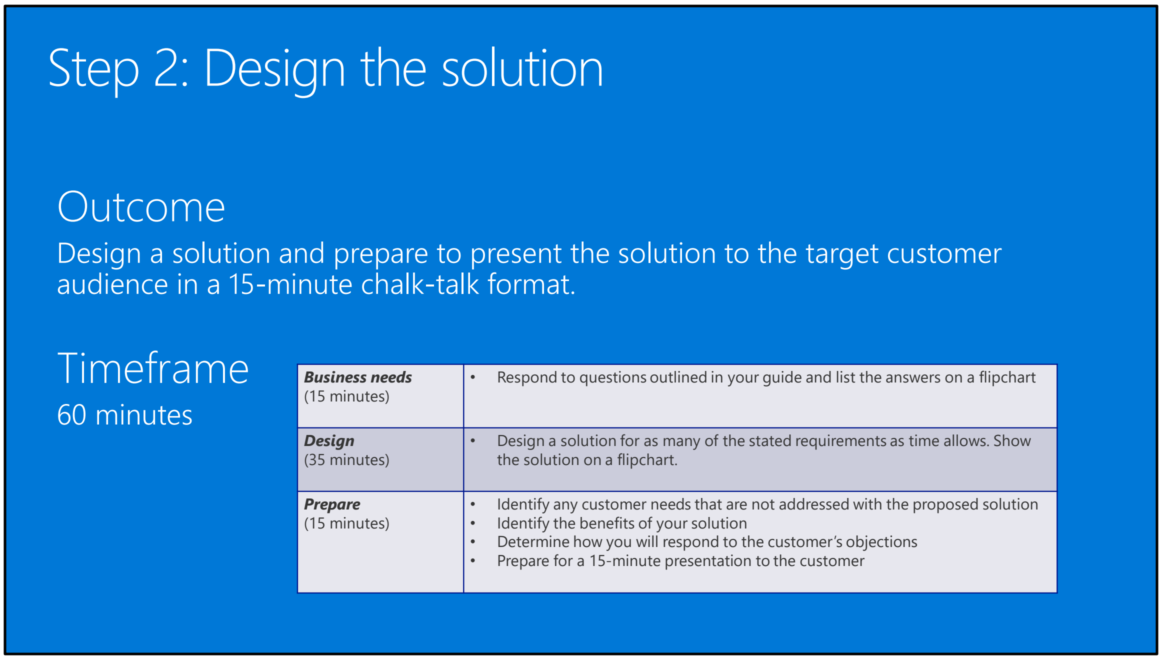


Common scenarios

Monolithic vs. Microservices – storage state


State in Monolithic approach
State in Microservices approach
stateless presentation services
stateless service
stateful services
stateless services with separate stores

This diagram presents a comparison of State storage in monolithic versus microservices approaches, from which you can draw inspiration. You will find this diagram within the Whiteboard Design Session Student Guide.



Step 2: Design the solution

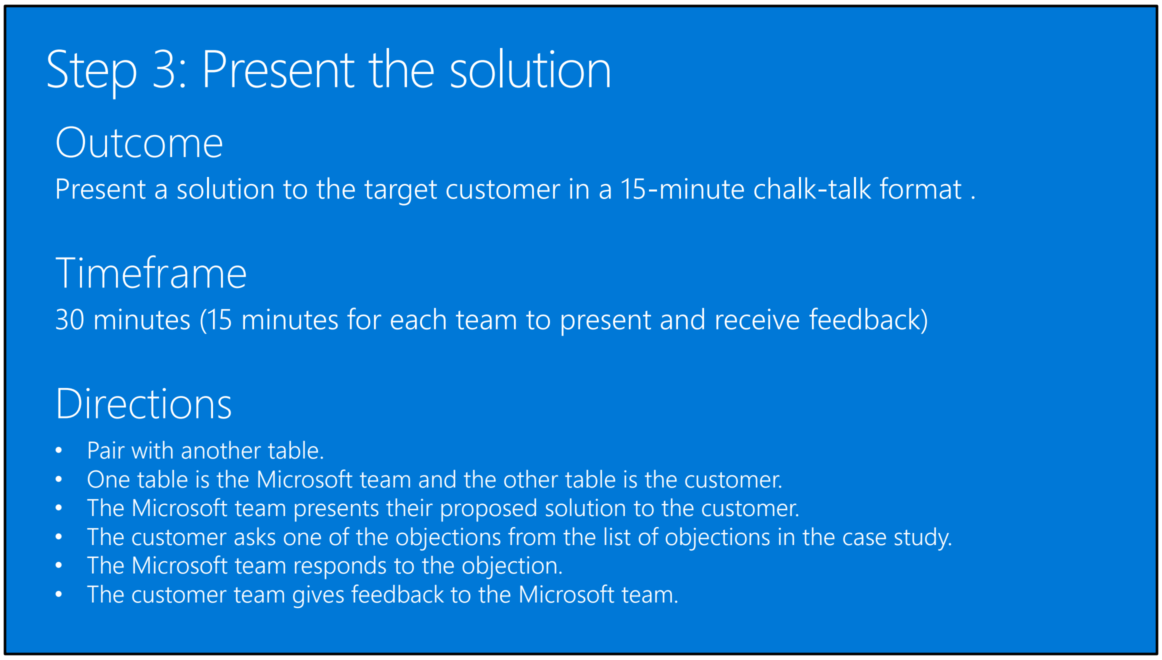
Outcome

Design a solution and prepare to present the solution to the target customer audience in a 15-minute chalk-talk format.

Timeframe

60 minutes

Business needs (15 minutes)	• Respond to questions outlined in your guide and list the answers on a flipchart
Design (35 minutes)	• Design a solution for as many of the stated requirements as time allows. Show the solution on a flipchart.
Prepare (15 minutes)	• Identify any customer needs that are not addressed with the proposed solution • Identify the benefits of your solution • Determine how you will respond to the customer's objections • Prepare for a 15-minute presentation to the customer



Step 3: Present the solution

Outcome

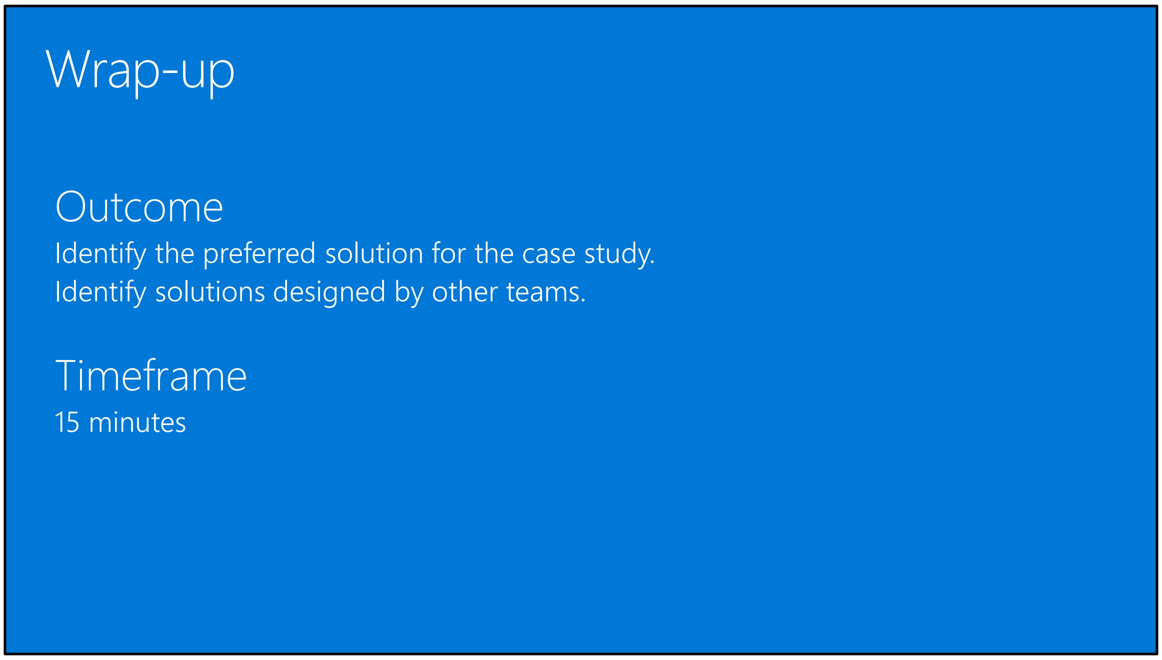
Present a solution to the target customer in a 15-minute chalk-talk format .

Timeframe

30 minutes (15 minutes for each team to present and receive feedback)

Directions

- Pair with another table.
- One table is the Microsoft team and the other table is the customer.
- The Microsoft team presents their proposed solution to the customer.
- The customer asks one of the objections from the list of objections in the case study.
- The Microsoft team responds to the objection.
- The customer team gives feedback to the Microsoft team.



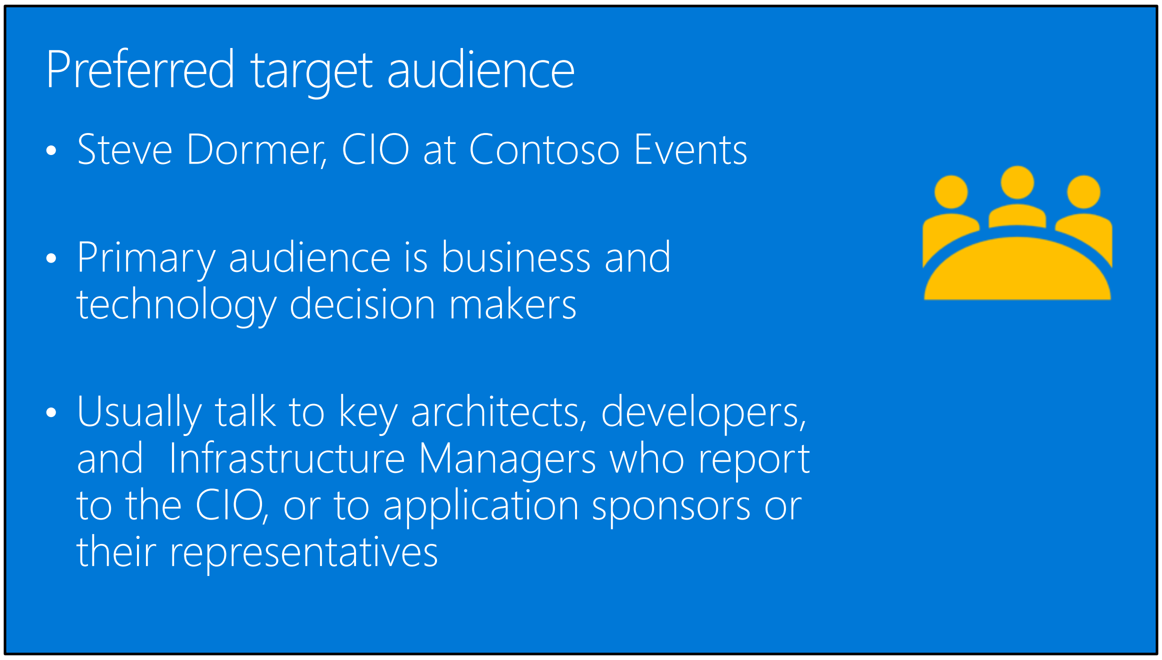
Wrap-up

Outcome

Identify the preferred solution for the case study.
Identify solutions designed by other teams.

Timeframe

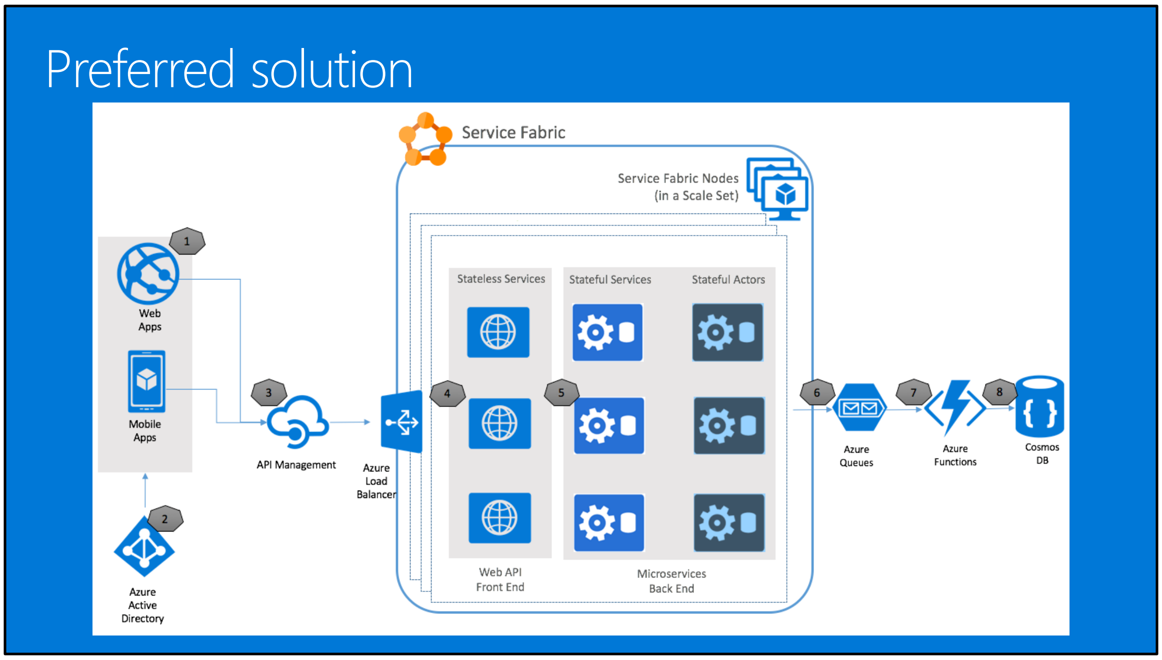
15 minutes



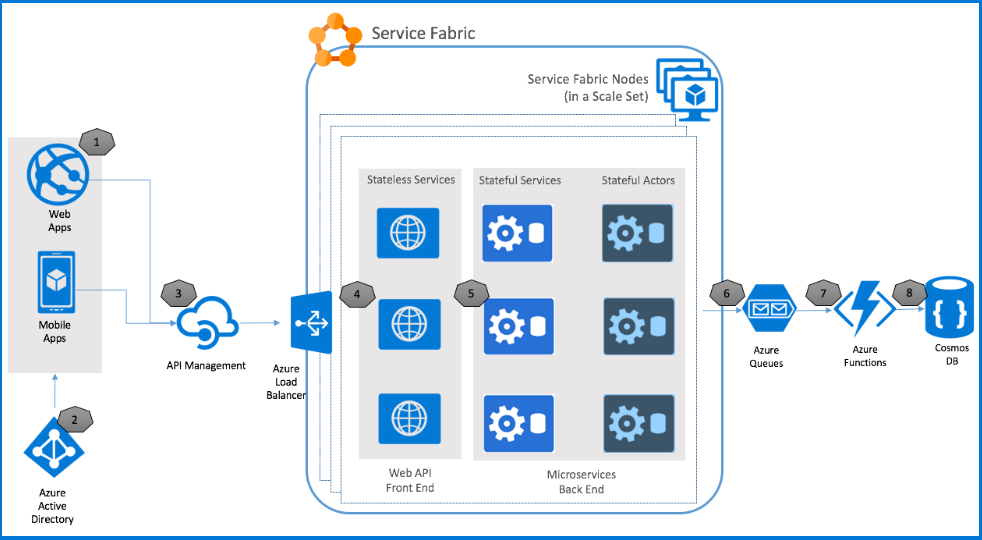
Preferred target audience

- Steve Dormer, CIO at Contoso Events
- Primary audience is business and technology decision makers
- Usually talk to key architects, developers, and Infrastructure Managers who report to the CIO, or to application sponsors or their representatives

- Steve Dormer, CIO at Contoso Events
- The primary audience is the technical strategic decision-maker with influential solution architects, or lead technical personnel in development or operations. For this example this could include the CIO and his core team.
- Usually we talk to the key architects, developers and infrastructure managers who report to the CIO, or to key solution sponsors or those that represent the business unit IT or developers that report to those sponsors.



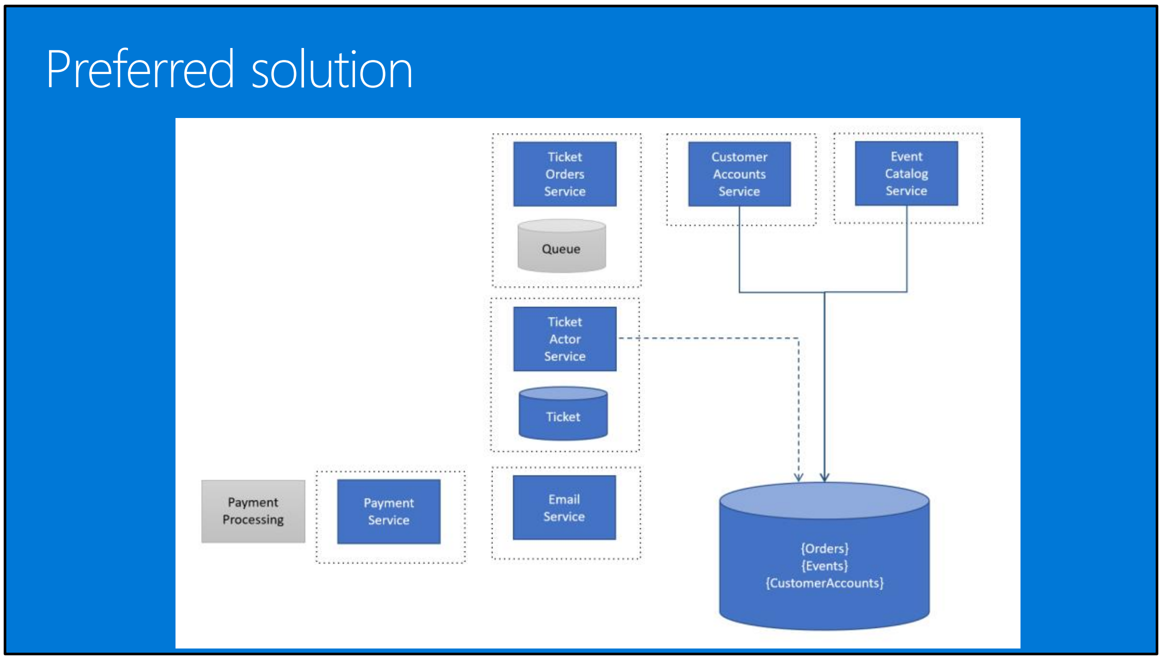
Preferred solution



This preferred solution is just one of many viable options

From a high-level:

- Starting with #1 on the left, Contoso Events will have both web and mobile applications that consume the back-end APIs for the solution.
- Users will authenticate to applications using tokens issued by Azure AD B2C.
- We will use an API Management layer as a gateway to all HTTP Web APIs exposed by the solution. It will be configured to authorize tokens issued by trusted Azure B2C tenants. This can be expanded to additional token issuers for third parties in future.
- Requests to HTTP Web APIs at the front end will go through Azure Load Balancer and distribute across the available Service Fabric nodes in the cluster.
- We will implement Business functionality through stateful services and actors, which will be called by Web APIs. These compose the microservices back end, which will sync their data back to the Cosmos DB instance for ad-hoc queries, by writing the jobs to an Azure queue.
- An Azure Function will handle processing the queue and updating the TicketOrders and related collections in Cosmos DB, according to business rules.



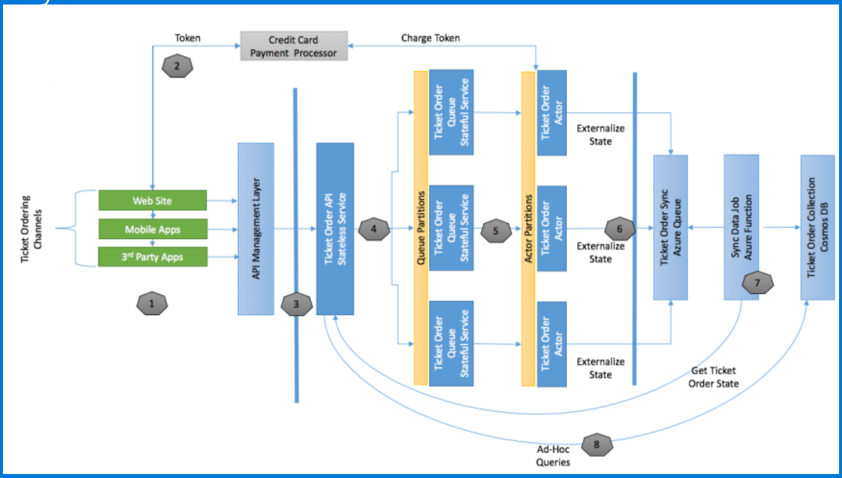
Preferred solution

Here is a high-level architecture of the core services that compose the new microservices architecture, as well as the state they hold.

- The Ticket Orders Service, on top of the diagram, will take advantage of reliable queues provided by Service Fabric to persist requests.
- The Ticket Actor Service processes orders in the queue, and represents a single instance of an order and its processing workflow and state.
- When an order is processed, the state is externalized for shared read only services to support aggregation across other data and to optimize reads.

Preferred solution

Scalability of ticket orders

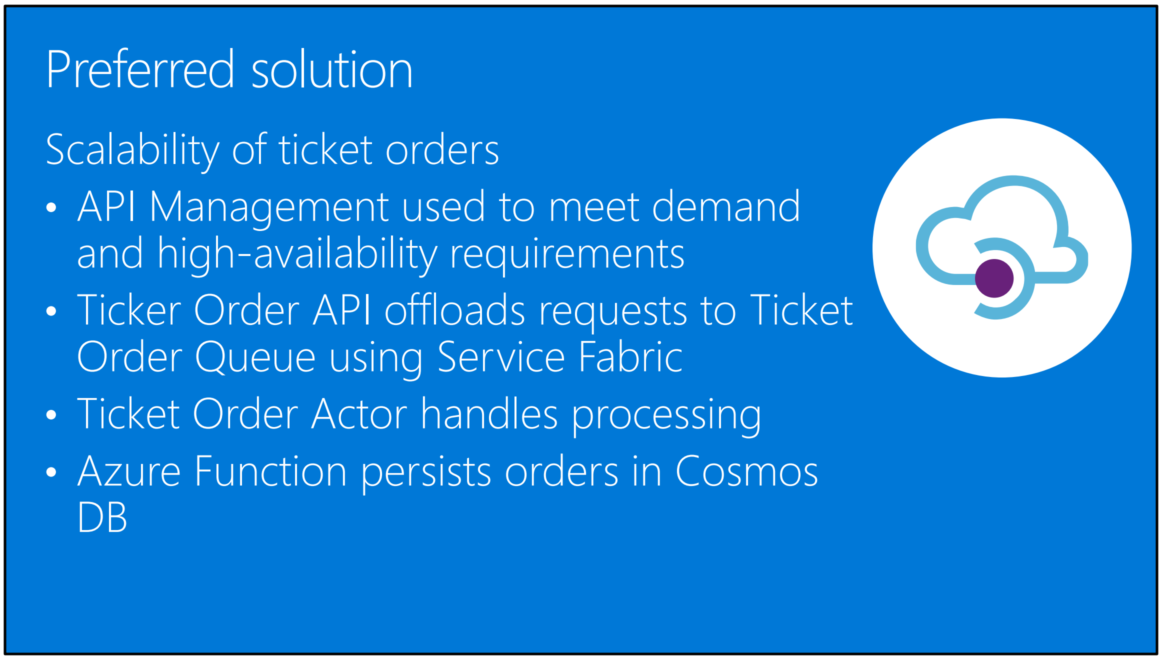


Illustrate in more detail the Service Fabric services and components participating in a ticket ordering request.

- This diagram illustrates the Service Fabric services and components participating in a ticket ordering request.
- The left-hand side of the diagram shows the various ticket ordering channels, such as the Contoso Events consumer web site and mobile applications, as well as third parties who build applications that place ticket orders.
- Following the numbered steps, the UI will pre-flight the credit card charge from the order page and supply the resulting token to the back-end processing of the order.
- All ticket order requests will call the Ticket Order API through the API Management layer, which will queue the request for processing, passing the token for credit card validation, order details, and any other information required to complete the request asynchronously.
- The Ticket Order Queue stateful service will process these requests by instantiating a Ticket Order Actor to handle the processing workflow. This actor is responsible for charging the credit card, finalizing the order, and notifying the customer to give them access to their order.
- The actor will also write to a Ticket Order Sync queue to offload sending order data

to Cosmos DB for reports, analytics, and related ad-hoc queries.

- An Azure Function will read from the queue and handle updates to Cosmos DB. In order to ensure the latest data is always persisted, the function will retrieve the latest order state and update the Cosmos DB Ticket Orders collection.
- Web APIs will expose data from Cosmos DB for additional ad-hoc queries against ticket orders.



Preferred solution

Scalability of ticket orders

- API Management used to meet demand and high-availability requirements
- Ticker Order API offloads requests to Ticket Order Queue using Service Fabric
- Ticket Order Actor handles processing
- Azure Function persists orders in Cosmos DB

Describe the scalability features of this design, including any partitioning strategies that are applicable.

- API Management Premium features support scaling and multi-region topologies to meet demand and high availability requirements.
- The stateless Ticket Order API offloads valid requests to the stateful Ticket Order Queue. This queue is partitioned by instance count so that requests can be distributed by Service Fabric to the appropriate node or service instance according to availability to process the message.
    - The stateful Ticket Order Queue is partitioned by instance count (from 1 to n).
    - When the stateless Ticket Order API offloads valid requests to this queue, those requests are randomly distributed by Service Fabric across these partitions, removing the bottleneck of writes to the queue, thanks to parallel distribution.
- The stateful Ticket Order Actor handles processing from this queue and given the number of parallel orders helps the solution to scale by maintaining the state of any number of parallel orders.
    - The actor is partitioned by order id, which allows for very fine-grained distribution of state, per order, across the cluster.

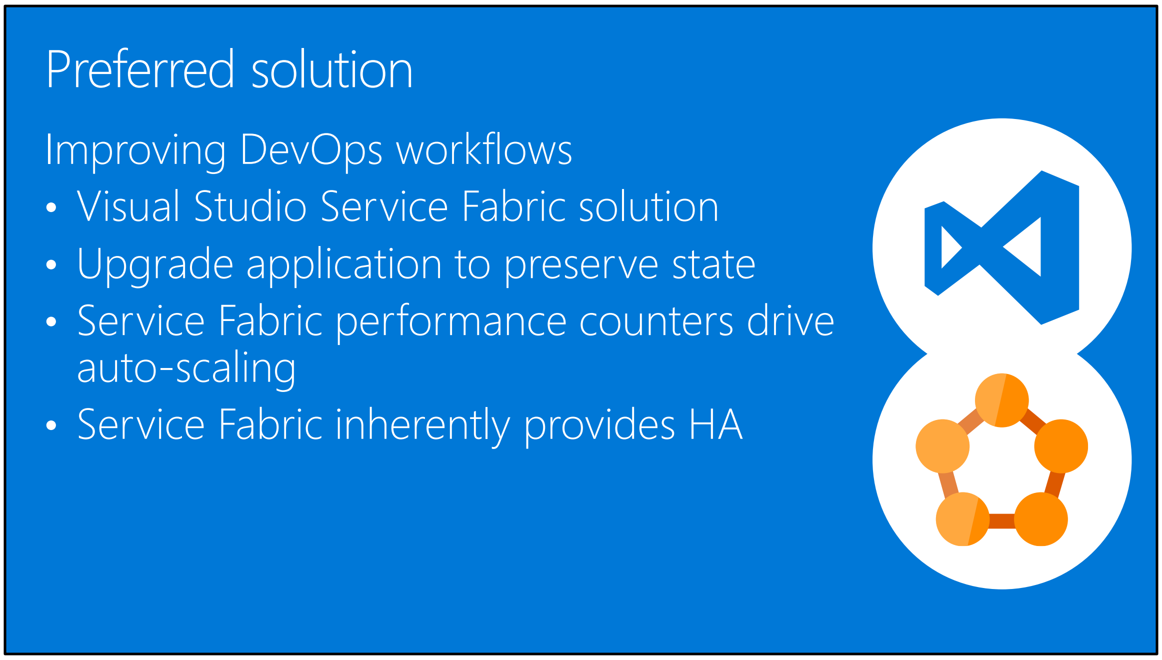
  - Millions of these can exist and distribute across the cluster and inactive actors are evicted from memory automatically to conserve resources.

Describe the resiliency of this use case. How can you create an asynchronous ticket order request and guarantee processing? Are there any potential points of failure? How will you address those?

- Web applications will not report success unless the Payment Processing succeeds and the order is successfully queued. The queue does not report successful receipt of the message until a quorum is reached.
- All Web API calls go through API Management, which can be scaled within a region, or deployed to multiple regions.
- The Ticket Order Actor does not remove the order from the queue unless it can successfully process its workflow and save to the Order Sync Queue.
- The Azure Function that processes the Order Sync Queue removes messages from the queue if processing is successful to Cosmos DB.
- Any of these messages that can’t be processed are moved to a poison queue, to be retried or processed again through another mechanism.
- For additional visibility into queue / function processing errors it is good practice to monitor queue sizes that pass a reasonable threshold of standard solution behavior.

Describe how you will enable external clients to reach stateless HTTP services exposed from the Azure load balancer.

- When you publish a stateless HTTP service the Service Fabric provides a relative URL to the service according to the configuration you supply.
- This way, multiple HTTP services can be deployed to a single node and still be uniquely addressable at port 80 or 443.
- In the case of this solution, the API Management layer will consume those endpoints and forward requests.



Preferred solution

Improving DevOps workflows

- Visual Studio Service Fabric solution
- Upgrade application to preserve state
- Service Fabric performance counters drive auto-scaling
- Service Fabric inherently provides HA

How would you structure the Visual Studio solution so that developers can run, debug and publish the entire solution, but also be able to publish and upgrade individual microservices (could be one or more service grouped together)?

- A typical Service Fabric solution has a top-level application that can be used to publish all services associated with it. To publish the entire suite of services in a solution you can add all services to this top-level application.
- You can create additional Service Fabric applications in the solution that isolate specific services for deployment.

Describe to the customer how they can upgrade services in situ and preserve state; handle rollback and roll forward; and service self-healing features.

- When an application is deployed, you can choose to “upgrade” the application. This preserves any state associated with stateful services if applicable. This will retire previous versions once they complete requests in process, while sending new requests to the new version of the service.
- If during the upgrade process there is a problem with the services being deployed, the upgrade is rolled back and the previous version of the services continue to operate.
- If the Service Fabric runtime detects any service instances are no longer

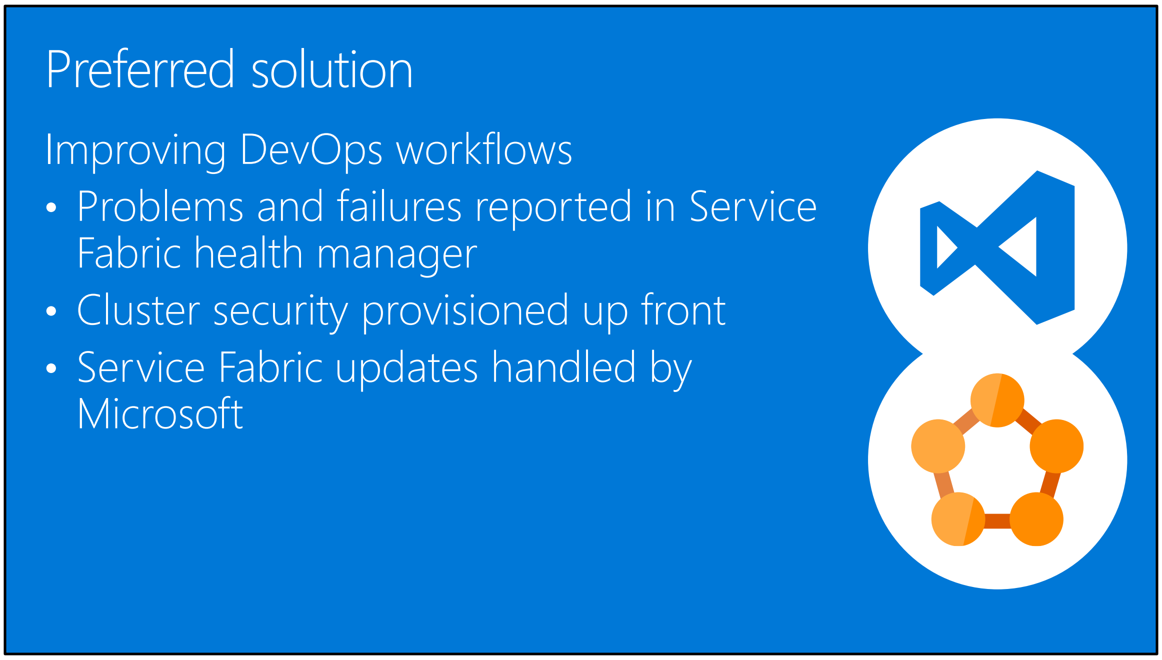
operational, new instances of the service are initialized to maintain the required minimum instances for the service.

Explain how the Service Fabric cluster handles auto-scaling. How does Service Fabric help the customer to make better utilization of their compute resources?

- Currently, performance counters emitted by the Service Fabric cluster drive auto-scaling. You can set up a base requirement for minimum nodes in the cluster (based on reliability level chosen) and configure auto-scale rules to scale up or down within that range, as the available nodes in the cluster become fully or less utilized.
- When you deploy services, they are distributed across the available nodes in the Service Fabric cluster – including replicas for stateful services. Service Fabric will ensure that services are distributed across nodes according to any placement constraints, while ensuring that nodes are densely populated.

How would you plan for high availability (HA) for Service Fabric in this solution?

- Service Fabric inherently provides high availability within the cluster region.
- You achieve multi-region high availability for disaster recovery scenarios by using the backup/restore capability of stateful services.
- You can also create real time high availability across regions by configuring Traffic Manager to route traffic to both regions. In this case, any stateful services should be capable of reloading their latest state from external storage if not present in the region's cluster.



Preferred solution

Improving DevOps workflows

- Problems and failures reported in Service Fabric health manager
- Cluster security provisioned up front
- Service Fabric updates handled by Microsoft

Explain to the customer how Service Fabric can help the customer have visibility into overall solution health.

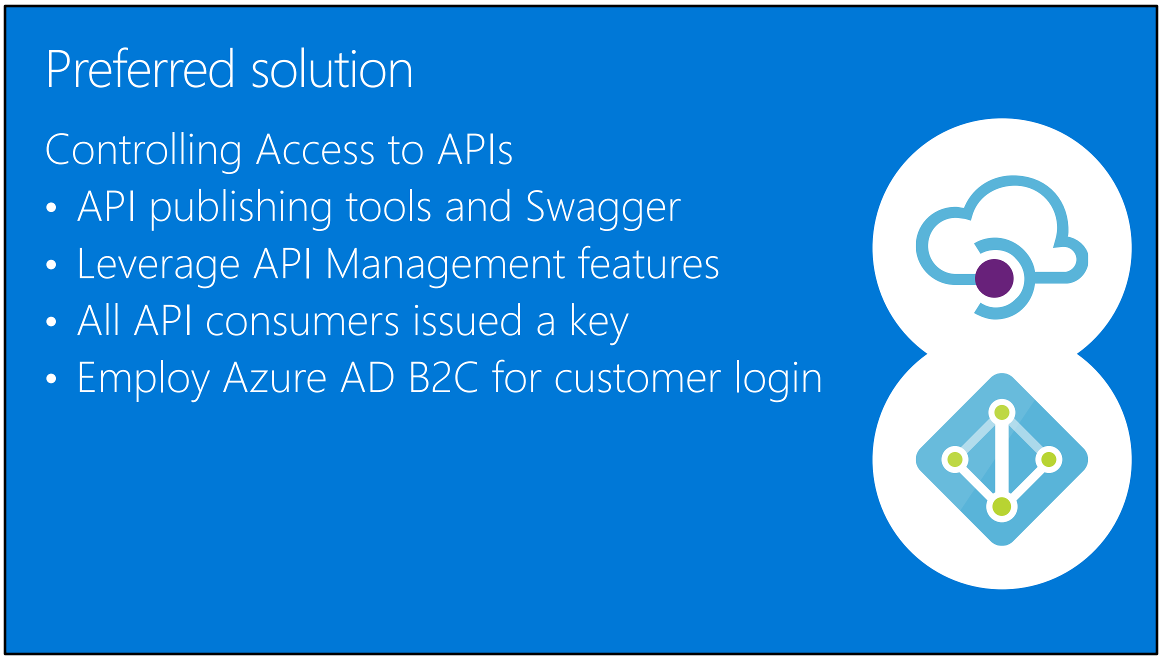
- If you report problems and failures to the Azure Service Fabric health manager from your service code, you can use standard health monitoring tools that Service Fabric provides to check the health status.
- For example, you can report configuration errors that would prevent the solution from reliable operation, and report these as exceptions to the Service Fabric.
- Exceptions that are detected while the Service Fabric tries to perform upgrades will trigger a rollback operation to avoid introducing this failure into the system.

How can you update cluster settings after the fact? What kind of settings might you want to update?

- You cannot change the security of the cluster after provisioning it, so it is important to set the cluster up as a secure cluster, from the beginning.
- You may need to modify the ARM template you used to create the cluster, add new ports you want to expose, add new node types for scale tiers, and then re-apply the template to safely upgrade the cluster nodes.

How will you keep your cluster up to date with the latest Service Fabric SDK?

- Microsoft updates the runtime SDK on all clusters, unless the cluster is in an unhealthy state. You do not have to update the cluster yourself to keep it current.



Preferred solution

Controlling Access to APIs

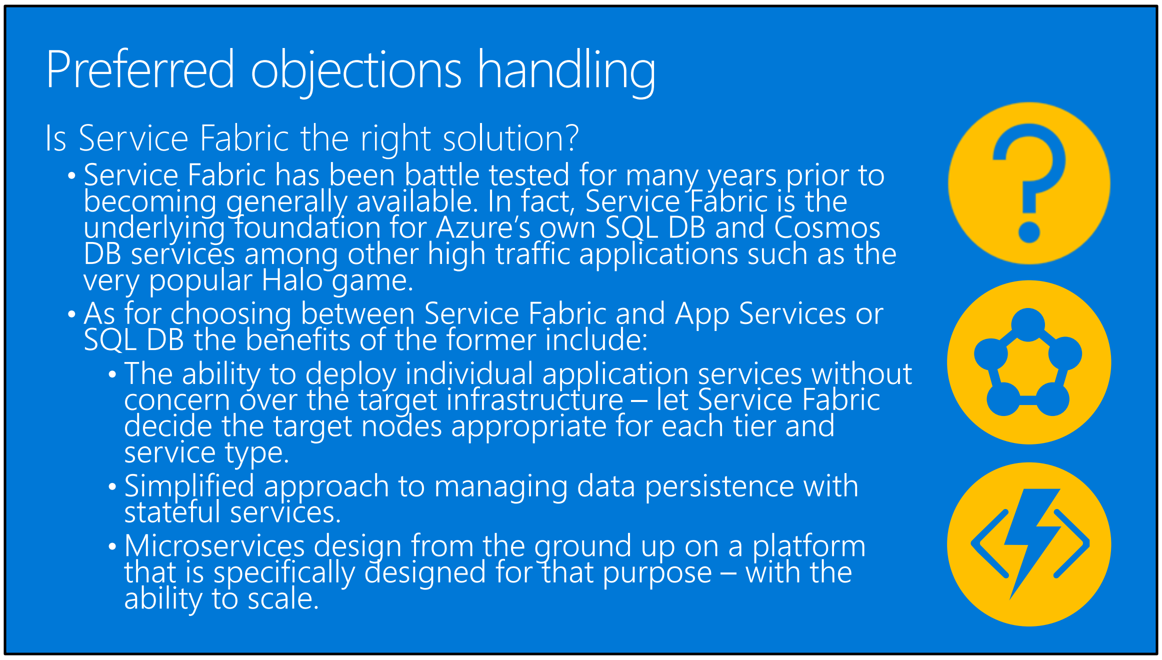
- API publishing tools and Swagger
- Leverage API Management features
- All API consumers issued a key
- Employ Azure AD B2C for customer login

Describe how API Management may be useful to control access to APIs exposed by the solution.

- Initially the solution will benefit from API publishing tools and swagger, API security policy and token validation and internal applications that can be created as pre-assigned API consumers without a sophisticated setup.
- Eventually, as the partner ecosystem is built out, the customer will want to leverage many more API Management features such as API consumer onboarding and policy management, API consumer self-service features such via the consumer portal, blog, API documentation, incident reporting tools and API consumer throttling and usage metrics reporting.

How would you identify the user and the API consumer or application?

- All consumers, including internal applications, will be issued a consumer key.
- The solution will employ Azure Active Directory B2C for customer login, and will use the same mechanism for API security. API Management policy will be set up to authorize access only to callers with a signed Azure AD token (JWT) issued by the solution tenant, for consumer applications. Corporate applications may introduce another Azure AD tenant that syncs with their internal AD, for example.



Preferred objections handling

Is Service Fabric the right solution?

- Service Fabric has been battle tested for many years prior to becoming generally available. In fact, Service Fabric is the underlying foundation for Azure's own SQL DB and Cosmos DB services among other high traffic applications such as the very popular Halo game.
- As for choosing between Service Fabric and App Services or SQL DB the benefits of the former include:
  - The ability to deploy individual application services without concern over the target infrastructure – let Service Fabric decide the target nodes appropriate for each tier and service type.
  - Simplified approach to managing data persistence with stateful services.
  - Microservices design from the ground up on a platform that is specifically designed for that purpose – with the ability to scale.

While we are interested in the microservices approach, we are still comparing Service Fabric with PaaS features such as App Services and SQL DB. How mature is Service Fabric by comparison?

- Service Fabric has been battle tested for many years prior to becoming generally available. In fact, Service Fabric is the underlying foundation for Azure's own SQL DB and Cosmos DB services among other high traffic applications such as the very popular Halo game.
-
- As for choosing between Service Fabric and App Services or SQL DB the benefits of the former include:
  - The ability to deploy individual application services without concern over the target infrastructure – let Service Fabric decide the target nodes appropriate for each tier and service type.
  - Simplified approach to managing data persistence with stateful services.
  - Microservices design from the ground up on a platform that is specifically designed for that purpose – with the ability to scale.

Microservices architectures are completely new to the Contoso Events team. If we were to go forward with Service Fabric, we would like to understand what skills the

team can carry forward, and how much of a learning curve exists.

- Service Fabric is a natural transition for .NET developers in many respects:
  - They can continue to use Visual Studio for development, debugging and publishing applications
  - They can leverage Service Fabric project templates to kick-start their understanding of Service Fabric services.
  - The programming model for services is familiar.
  - Working with stateful services is also familiar in the sense that state is defined via objects (POCO) and serialized as part of the service implementation.

We'd like to understand if stateful services or stateful actors will help us with ticket ordering throughput, workflow and state management, and easier rollouts of changes to this process.

- Stateful services are backed by robust and reliable storage. When data (state) is saved by the service, it is not confirmed (committed) unless a quorum is reached.
- By using the stateful actor, not only is the persistence of the actual ticket order handled by the Service Fabric at scale, but the actor can be wholly responsible for the workflow required to complete the order
- When updates to this actor are required, the existing state is preserved, any active instances can continue completing their work, and the new actor functionality or state requirements can be rolled out safely across the nodes in the cluster, eventually retiring the previous version.

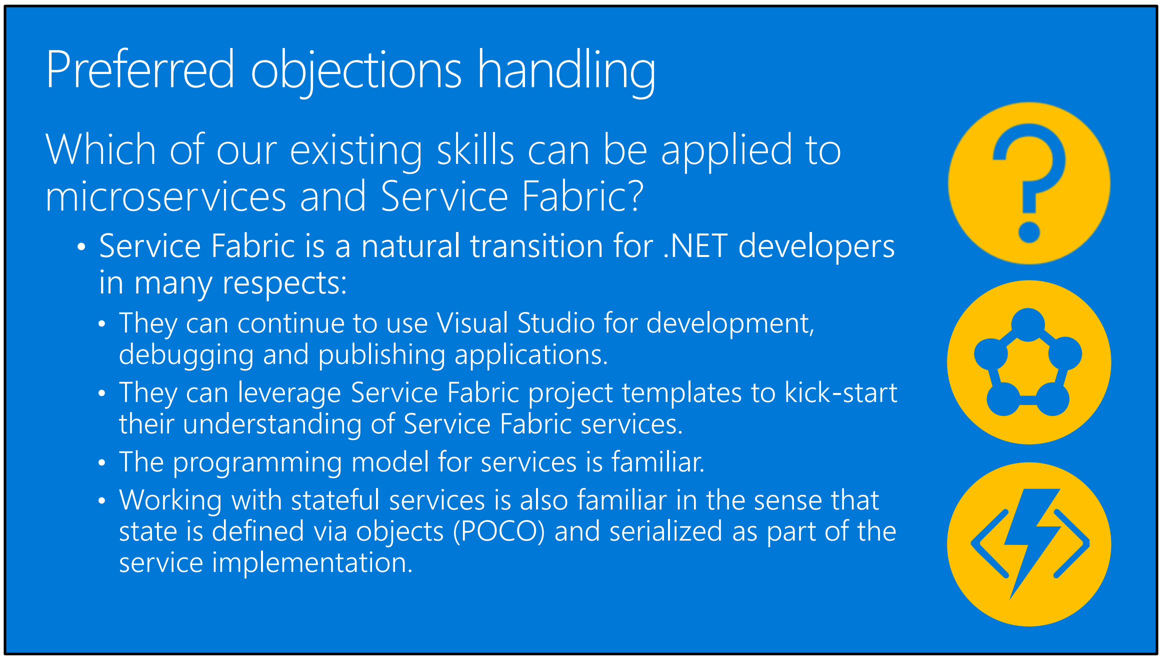
We are not clear how and where to incorporate stateful services and actors alongside other storage such as Cosmos DB. We need the ability to support robust ad-hoc queries against our system data such as events, customers, orders and related metrics – but would like to take advantage of the performance and reliability of Service Fabric stateful options as well.

- Stateful services make it easy to save and retrieve state, and distribute that state for higher availability by using a partitioning strategy. Each partition has its own replica set for reliability.
- You can replicate this state to an external store like Cosmos DB to support ad-hoc querying, analytics and disaster recovery.

Could we consider Azure Functions as an alternative back end implementation for our APIs?

- While it is possible to create functions that run behind API Management endpoints, they are best employed for decoupled, asynchronous background operations that can be run at scale without concern for the specific server running that operation.
- In this solution, Azure Functions allowed for decoupling the external storage

location of orders, without the need to update Service Fabric configurations on change. It also allowed for a separate scale-out tier for that work.
- In a solution such as a mobile application back end, functions could be useful if they don’t need to comingle with other solution aspects – such as acting as their own microservice with a targeted purpose.



Preferred objections handling

Which of our existing skills can be applied to microservices and Service Fabric?

- Service Fabric is a natural transition for .NET developers in many respects:
  - They can continue to use Visual Studio for development, debugging and publishing applications.
  - They can leverage Service Fabric project templates to kick-start their understanding of Service Fabric services.
  - The programming model for services is familiar.
  - Working with stateful services is also familiar in the sense that state is defined via objects (POCO) and serialized as part of the service implementation.

While we are interested in the microservices approach, we are still comparing Service Fabric with PaaS features such as App Services and SQL DB. How mature is Service Fabric by comparison?

- Service Fabric has been battle tested for many years prior to becoming generally available. In fact, Service Fabric is the underlying foundation for Azure's own SQL DB and Cosmos DB services among other high traffic applications such as the very popular Halo game.
-
- As for choosing between Service Fabric and App Services or SQL DB the benefits of the former include:
  - The ability to deploy individual application services without concern over the target infrastructure – let Service Fabric decide the target nodes appropriate for each tier and service type.
  - Simplified approach to managing data persistence with stateful services.
  - Microservices design from the ground up on a platform that is specifically designed for that purpose – with the ability to scale.

Microservices architectures are completely new to the Contoso Events team. If we were to go forward with Service Fabric, we would like to understand what skills the

team can carry forward, and how much of a learning curve exists.

- Service Fabric is a natural transition for .NET developers in many respects:
    - They can continue to use Visual Studio for development, debugging and publishing applications
    - They can leverage Service Fabric project templates to kick-start their understanding of Service Fabric services.
    - The programming model for services is familiar.
    - Working with stateful services is also familiar in the sense that state is defined via objects (POCO) and serialized as part of the service implementation.

We'd like to understand if stateful services or stateful actors will help us with ticket ordering throughput, workflow and state management, and easier rollouts of changes to this process.

- Stateful services are backed by robust and reliable storage. When data (state) is saved by the service, it is not confirmed (committed) unless a quorum is reached.
- By using the stateful actor, not only is the persistence of the actual ticket order handled by the Service Fabric at scale, but the actor can be wholly responsible for the workflow required to complete the order
- When updates to this actor are required, the existing state is preserved, any active instances can continue completing their work, and the new actor functionality or state requirements can be rolled out safely across the nodes in the cluster, eventually retiring the previous version.

We are not clear how and where to incorporate stateful services and actors alongside other storage such as Cosmos DB. We need the ability to support robust ad-hoc queries against our system data such as events, customers, orders and related metrics – but would like to take advantage of the performance and reliability of Service Fabric stateful options as well.

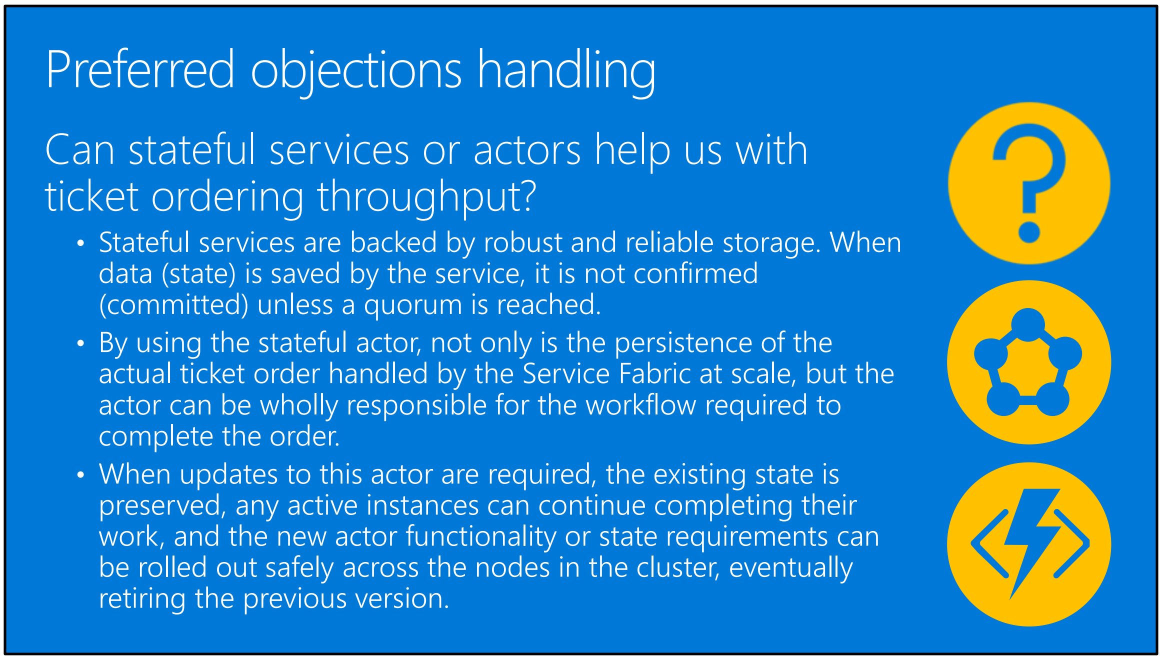
- Stateful services make it easy to save and retrieve state, and distribute that state for higher availability by using a partitioning strategy. Each partition has its own replica set for reliability.
- You can replicate this state to an external store like Cosmos DB to support ad-hoc querying, analytics and disaster recovery.

Could we consider Azure Functions as an alternative back end implementation for our APIs?

- While it is possible to create functions that run behind API Management endpoints, they are best employed for decoupled, asynchronous background operations that can be run at scale without concern for the specific server running that operation.
- In this solution, Azure Functions allowed for decoupling the external storage

location of orders, without the need to update Service Fabric configurations on change. It also allowed for a separate scale-out tier for that work.

- In a solution such as a mobile application back end, functions could be useful if they don’t need to comingle with other solution aspects – such as acting as their own microservice with a targeted purpose.



Preferred objections handling

Can stateful services or actors help us with ticket ordering throughput?

- Stateful services are backed by robust and reliable storage. When data (state) is saved by the service, it is not confirmed (committed) unless a quorum is reached.
- By using the stateful actor, not only is the persistence of the actual ticket order handled by the Service Fabric at scale, but the actor can be wholly responsible for the workflow required to complete the order.
- When updates to this actor are required, the existing state is preserved, any active instances can continue completing their work, and the new actor functionality or state requirements can be rolled out safely across the nodes in the cluster, eventually retiring the previous version.

While we are interested in the microservices approach, we are still comparing Service Fabric with PaaS features such as App Services and SQL DB. How mature is Service Fabric by comparison?

- Service Fabric has been battle tested for many years prior to becoming generally available. In fact, Service Fabric is the underlying foundation for Azure's own SQL DB and Cosmos DB services among other high traffic applications such as the very popular Halo game.
-
- As for choosing between Service Fabric and App Services or SQL DB the benefits of the former include:
    - The ability to deploy individual application services without concern over the target infrastructure – let Service Fabric decide the target nodes appropriate for each tier and service type.
    - Simplified approach to managing data persistence with stateful services.
    - Microservices design from the ground up on a platform that is specifically designed for that purpose – with the ability to scale.

Microservices architectures are completely new to the Contoso Events team. If we were to go forward with Service Fabric, we would like to understand what skills the

team can carry forward, and how much of a learning curve exists.

- Service Fabric is a natural transition for .NET developers in many respects:
  - They can continue to use Visual Studio for development, debugging and publishing applications
  - They can leverage Service Fabric project templates to kick-start their understanding of Service Fabric services.
  - The programming model for services is familiar.
  - Working with stateful services is also familiar in the sense that state is defined via objects (POCO) and serialized as part of the service implementation.

We'd like to understand if stateful services or stateful actors will help us with ticket ordering throughput, workflow and state management, and easier rollouts of changes to this process.

- Stateful services are backed by robust and reliable storage. When data (state) is saved by the service, it is not confirmed (committed) unless a quorum is reached.
- By using the stateful actor, not only is the persistence of the actual ticket order handled by the Service Fabric at scale, but the actor can be wholly responsible for the workflow required to complete the order
- When updates to this actor are required, the existing state is preserved, any active instances can continue completing their work, and the new actor functionality or state requirements can be rolled out safely across the nodes in the cluster, eventually retiring the previous version.

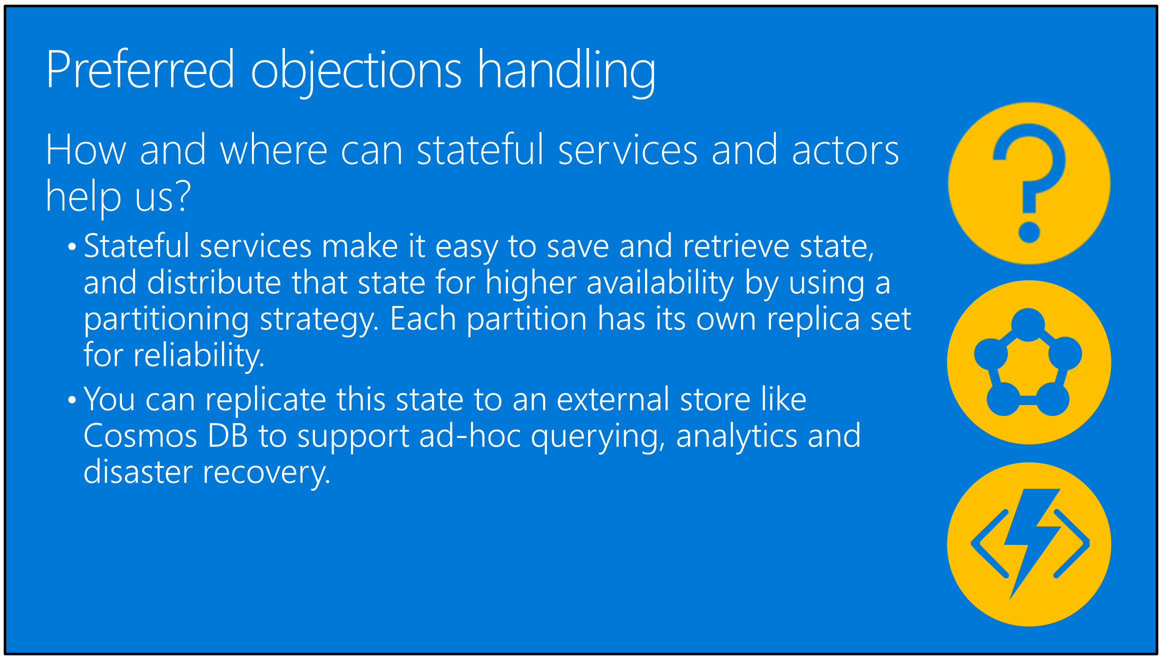
We are not clear how and where to incorporate stateful services and actors alongside other storage such as Cosmos DB. We need the ability to support robust ad-hoc queries against our system data such as events, customers, orders and related metrics – but would like to take advantage of the performance and reliability of Service Fabric stateful options as well.

- Stateful services make it easy to save and retrieve state, and distribute that state for higher availability by using a partitioning strategy. Each partition has its own replica set for reliability.
- You can replicate this state to an external store like Cosmos DB to support ad-hoc querying, analytics and disaster recovery.

Could we consider Azure Functions as an alternative back end implementation for our APIs?

- While it is possible to create functions that run behind API Management endpoints, they are best employed for decoupled, asynchronous background operations that can be run at scale without concern for the specific server running that operation.
- In this solution, Azure Functions allowed for decoupling the external storage

location of orders, without the need to update Service Fabric configurations on change. It also allowed for a separate scale-out tier for that work.
- In a solution such as a mobile application back end, functions could be useful if they don’t need to comingle with other solution aspects – such as acting as their own microservice with a targeted purpose.



Preferred objections handling

How and where can stateful services and actors help us?

- Stateful services make it easy to save and retrieve state, and distribute that state for higher availability by using a partitioning strategy. Each partition has its own replica set for reliability.
- You can replicate this state to an external store like Cosmos DB to support ad-hoc querying, analytics and disaster recovery.

While we are interested in the microservices approach, we are still comparing Service Fabric with PaaS features such as App Services and SQL DB. How mature is Service Fabric by comparison?

- Service Fabric has been battle tested for many years prior to becoming generally available. In fact, Service Fabric is the underlying foundation for Azure's own SQL DB and Cosmos DB services among other high traffic applications such as the very popular Halo game.
-
- As for choosing between Service Fabric and App Services or SQL DB the benefits of the former include:
    - The ability to deploy individual application services without concern over the target infrastructure – let Service Fabric decide the target nodes appropriate for each tier and service type.
    - Simplified approach to managing data persistence with stateful services.
    - Microservices design from the ground up on a platform that is specifically designed for that purpose – with the ability to scale.

Microservices architectures are completely new to the Contoso Events team. If we were to go forward with Service Fabric, we would like to understand what skills the

team can carry forward, and how much of a learning curve exists.

- Service Fabric is a natural transition for .NET developers in many respects:
  - They can continue to use Visual Studio for development, debugging and publishing applications
  - They can leverage Service Fabric project templates to kick-start their understanding of Service Fabric services.
  - The programming model for services is familiar.
  - Working with stateful services is also familiar in the sense that state is defined via objects (POCO) and serialized as part of the service implementation.

We'd like to understand if stateful services or stateful actors will help us with ticket ordering throughput, workflow and state management, and easier rollouts of changes to this process.

- Stateful services are backed by robust and reliable storage. When data (state) is saved by the service, it is not confirmed (committed) unless a quorum is reached.
- By using the stateful actor, not only is the persistence of the actual ticket order handled by the Service Fabric at scale, but the actor can be wholly responsible for the workflow required to complete the order
- When updates to this actor are required, the existing state is preserved, any active instances can continue completing their work, and the new actor functionality or state requirements can be rolled out safely across the nodes in the cluster, eventually retiring the previous version.

We are not clear how and where to incorporate stateful services and actors alongside other storage such as Cosmos DB. We need the ability to support robust ad-hoc queries against our system data such as events, customers, orders and related metrics – but would like to take advantage of the performance and reliability of Service Fabric stateful options as well.

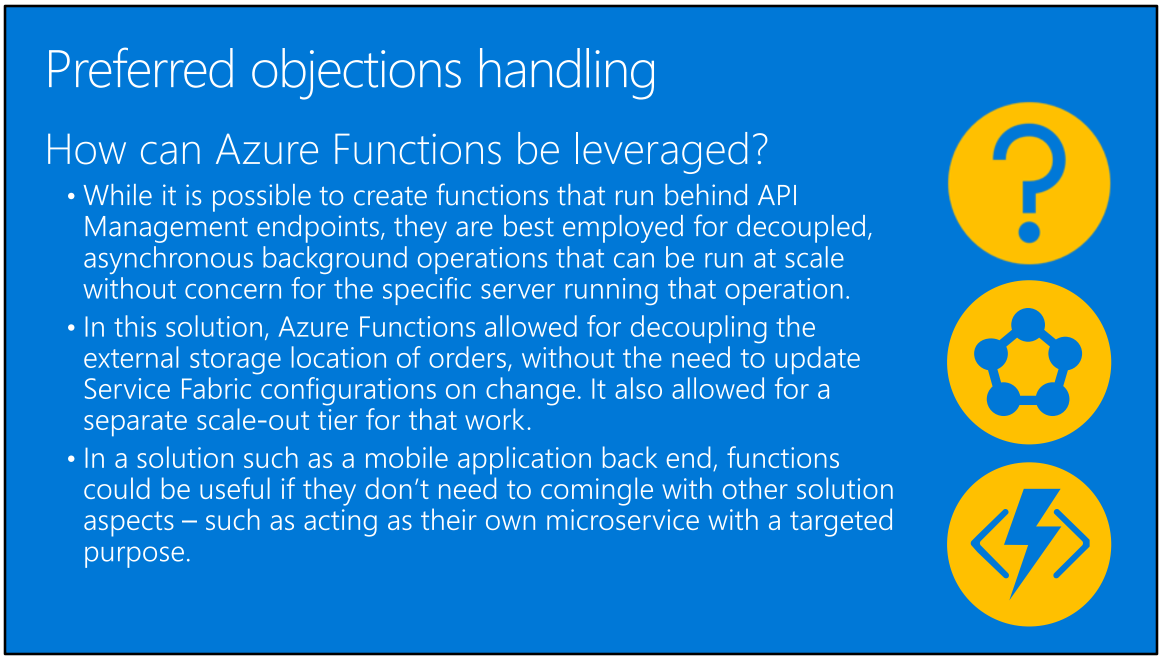
- Stateful services make it easy to save and retrieve state, and distribute that state for higher availability by using a partitioning strategy. Each partition has its own replica set for reliability.
- You can replicate this state to an external store like Cosmos DB to support ad-hoc querying, analytics and disaster recovery.

Could we consider Azure Functions as an alternative back end implementation for our APIs?

- While it is possible to create functions that run behind API Management endpoints, they are best employed for decoupled, asynchronous background operations that can be run at scale without concern for the specific server running that operation.
- In this solution, Azure Functions allowed for decoupling the external storage

location of orders, without the need to update Service Fabric configurations on change. It also allowed for a separate scale-out tier for that work.

- In a solution such as a mobile application back end, functions could be useful if they don’t need to comingle with other solution aspects – such as acting as their own microservice with a targeted purpose.



Preferred objections handling

How can Azure Functions be leveraged?

- While it is possible to create functions that run behind API Management endpoints, they are best employed for decoupled, asynchronous background operations that can be run at scale without concern for the specific server running that operation.
- In this solution, Azure Functions allowed for decoupling the external storage location of orders, without the need to update Service Fabric configurations on change. It also allowed for a separate scale-out tier for that work.
- In a solution such as a mobile application back end, functions could be useful if they don't need to comingle with other solution aspects – such as acting as their own microservice with a targeted purpose.

While we are interested in the microservices approach, we are still comparing Service Fabric with PaaS features such as App Services and SQL DB. How mature is Service Fabric by comparison?

- Service Fabric has been battle tested for many years prior to becoming generally available. In fact, Service Fabric is the underlying foundation for Azure's own SQL DB and Cosmos DB services among other high traffic applications such as the very popular Halo game.
-
- As for choosing between Service Fabric and App Services or SQL DB the benefits of the former include:
    - The ability to deploy individual application services without concern over the target infrastructure – let Service Fabric decide the target nodes appropriate for each tier and service type.
    - Simplified approach to managing data persistence with stateful services.
    - Microservices design from the ground up on a platform that is specifically designed for that purpose – with the ability to scale.

Microservices architectures are completely new to the Contoso Events team. If we were to go forward with Service Fabric, we would like to understand what skills the

team can carry forward, and how much of a learning curve exists.

- Service Fabric is a natural transition for .NET developers in many respects:
    - They can continue to use Visual Studio for development, debugging and publishing applications
    - They can leverage Service Fabric project templates to kick-start their understanding of Service Fabric services.
    - The programming model for services is familiar.
    - Working with stateful services is also familiar in the sense that state is defined via objects (POCO) and serialized as part of the service implementation.

We'd like to understand if stateful services or stateful actors will help us with ticket ordering throughput, workflow and state management, and easier rollouts of changes to this process.

- Stateful services are backed by robust and reliable storage. When data (state) is saved by the service, it is not confirmed (committed) unless a quorum is reached.
- By using the stateful actor, not only is the persistence of the actual ticket order handled by the Service Fabric at scale, but the actor can be wholly responsible for the workflow required to complete the order
- When updates to this actor are required, the existing state is preserved, any active instances can continue completing their work, and the new actor functionality or state requirements can be rolled out safely across the nodes in the cluster, eventually retiring the previous version.

We are not clear how and where to incorporate stateful services and actors alongside other storage such as Cosmos DB. We need the ability to support robust ad-hoc queries against our system data such as events, customers, orders and related metrics – but would like to take advantage of the performance and reliability of Service Fabric stateful options as well.

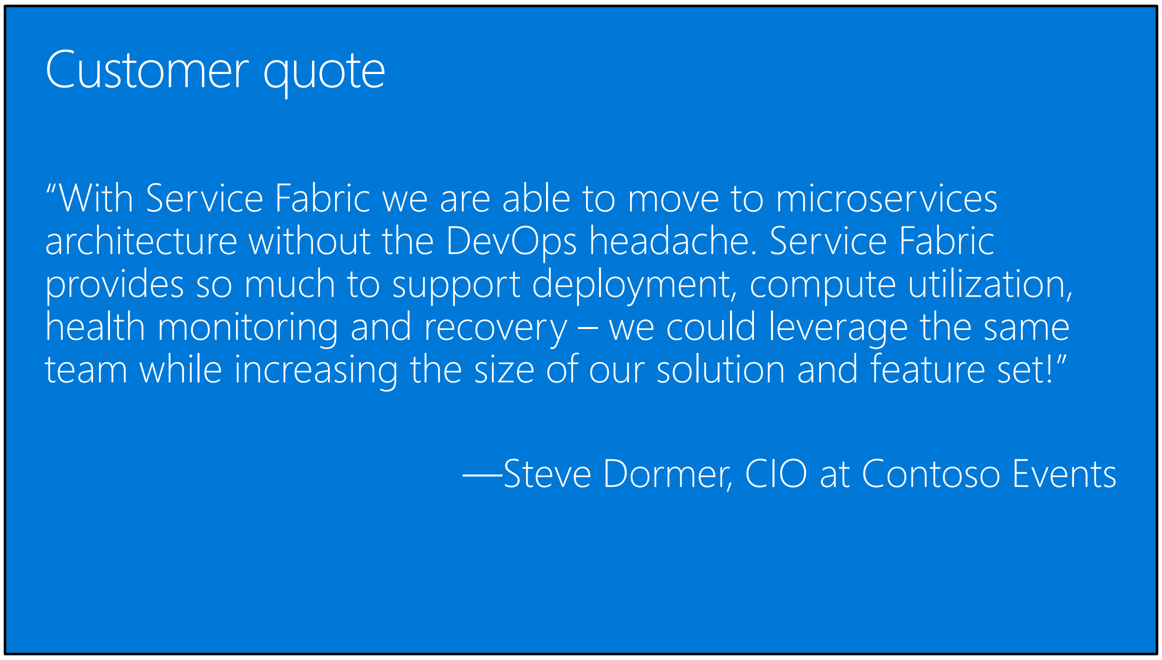
- Stateful services make it easy to save and retrieve state, and distribute that state for higher availability by using a partitioning strategy. Each partition has its own replica set for reliability.
- You can replicate this state to an external store like Cosmos DB to support ad-hoc querying, analytics and disaster recovery.

Could we consider Azure Functions as an alternative back end implementation for our APIs?

- While it is possible to create functions that run behind API Management endpoints, they are best employed for decoupled, asynchronous background operations that can be run at scale without concern for the specific server running that operation.
- In this solution, Azure Functions allowed for decoupling the external storage

location of orders, without the need to update Service Fabric configurations on change. It also allowed for a separate scale-out tier for that work.

- In a solution such as a mobile application back end, functions could be useful if they don’t need to comingle with other solution aspects – such as acting as their own microservice with a targeted purpose.



Customer quote

“With Service Fabric we are able to move to microservices architecture without the DevOps headache. Service Fabric provides so much to support deployment, compute utilization, health monitoring and recovery – we could leverage the same team while increasing the size of our solution and feature set!”

—Steve Dormer, CIO at Contoso Events

Microsoft