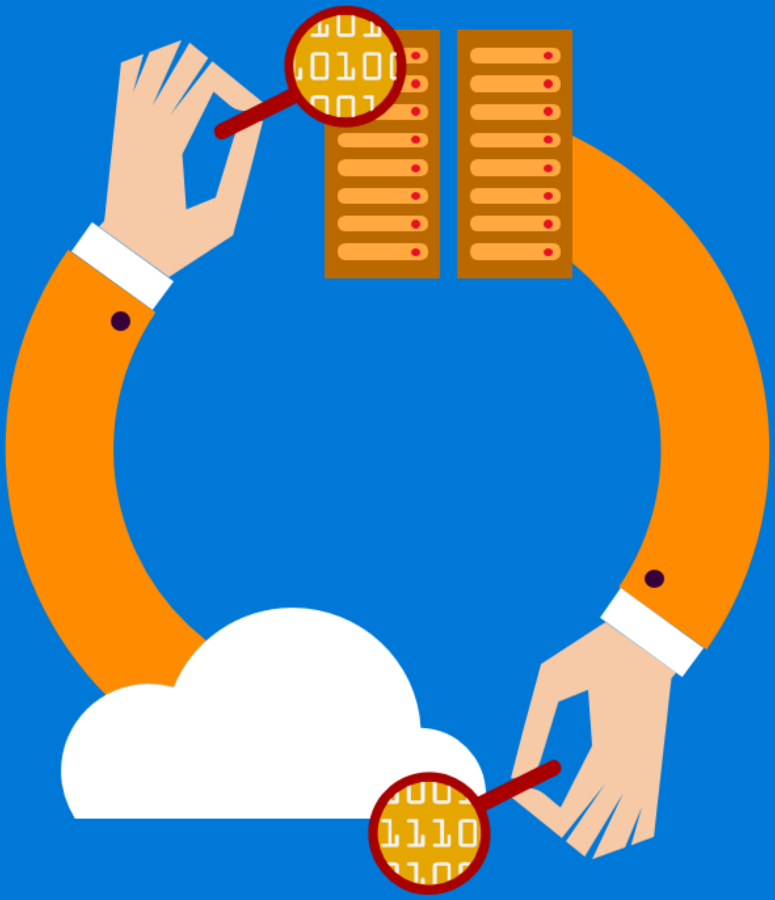
Microsoft



Design considerations for hybrid applications

By Marc van Eijk
AzureCAT

September 2018

Contents

Authored by Marc van Eijk. Edited by Bruce Hamilton. Reviewed by Damir Bersinic, Gavin Kemp, Daniel Neumann, and Emmanuel Sache.

Overview

Microsoft Azure is the only consistent hybrid cloud. It allows you to reuse your development investments and enables applications that can span global Azure, the sovereign Azure clouds, and Azure Stack, which is an extension of Azure in your datacenter. Applications that span clouds are also referred to as *hybrid applications*.

The Azure Application Architecture Guide describes a structured approach for designing applications that are scalable, resilient, and highly available. The considerations described in the Azure Application Architecture Guide equally apply to applications that are designed for a single cloud and for applications that span clouds.

This article augments the Pillars of software quality discussed in the Azure Application Architecture Guide, focusing specifically on designing hybrid applications. In addition, we add a *placement* pillar as hybrid applications are not exclusive to one cloud or one on-premises datacenter.

Hybrid scenarios vary greatly with the resources that are available for development, and span considerations such as geography, security, Internet access, and other considerations. Although this guide cannot enumerate your specific considerations, it can provide some key guidelines and best practices for you to follow. Successfully designing, configuring, deploying, and maintaining a hybrid application architecture involves many design considerations that might not be inherently known to you.

This document aims to aggregate the possible questions that might arise when implementing hybrid applications and provides considerations (these pillars) and best practices to work with them. By addressing these questions during the design phase, you'll avoid the issues they could cause in production.

Essentially, these are questions you need to think about before creating a hybrid application. To get started, you need to do the following:

- Identify and evaluate the application components.
- Evaluate application components against the pillars.

Identify and evaluate the application components

Each component of an application has its own specific role within the larger application and should be reviewed with all design considerations. Each component's requirements and features should map to these considerations to help determine the application architecture.

Decompose your application into its components by studying your application's architecture and determining what it consists of. Components can also include other applications that your application interacts with. As you identify the components, evaluate your intended hybrid operations according to their characteristics, such as the following:

- What is the purpose of the component?
- What are the interdependencies between the components?

For example, an application can have a front-end and back-end defined as two components. In a hybrid scenario, the front-end is in one cloud and the back-end is in the other. The application provides communication channels between the front-end and the user, and also between the front-end and the back-end.

An application component is defined by many forms and scenarios. The most important task is identifying them and their cloud or on-premises location.

The common application components to include in your inventory are listed in Table 1.

Table 1. Common application components

Component	Hybrid application guidance
Client connections	Your application (on any device) can access users in various ways, from a single-entry point, including the following: ▪ A client-server model that requires the user to have a client installed to work with the application. A server-based application that is accessed from a web browser. ▪ Client connections can include notifications when the connection is broken or alerts when roaming charges may apply.
Authentication	Authentication can be required for a user connecting to the application, or from one component connecting to another.
APIs	You can provide developers with programmatic access to your application with API sets and class libraries and provide a connection interface based on Internet standards. You can also use APIs to decompose an application into independently operating logical units.
Services	You can employ succinct services to provide the features for an application. A service can be the engine that the application runs on.
Queues	You can use queues to organize the status of the life cycles and states of your application's components. These queues can provide messaging, notifications, and buffering capabilities to subscribing parties.
Data storage	An application can be stateless or stateful. Stateful applications need data storage that can be met by numerous formats and volumes.
Data caching	A data caching component in your design can strategically address latency issues and play a role in triggering cloud bursting.
Data ingestion	Data can be submitted to an application in many ways, ranging from user-submitted values in a web form to continuously high-volume data flow.
Data processing	Your data processing tasks (such as reports, analytics, batch exports, and data transformation) can either be processed at the source or offloaded on a separate component using a copy of the data.

Evaluate application components against the pillars

For each component, evaluate its characteristics for each pillar. As you evaluate each component with all of the pillars, questions you might not have considered may become known to you that affect the design of the hybrid application. Acting on these considerations could add value in optimizing your application. Table 2 provides a description of each pillar as it relates to hybrid applications.

Table 2. Pillars

Pillar	Description
Placement	The strategic positioning of components in hybrid applications.
Scalability	The ability of a system to handle increased load.
Availability	The proportion of time that a hybrid application is functional and working.
Resiliency	The ability for a hybrid application to recover.
Manageability	Operations processes that keep a system running in production.
Security	Protecting hybrid applications and data from threats.

Placement

A hybrid application inherently has a placement consideration, such as for the datacenter.

Placement is the important task of positioning components so that they can best service a hybrid application. By definition, hybrid applications span locations, such as from on-premises to the cloud and among different clouds. You can place components of the application on clouds in two ways:

- **Vertical hybrid applications**
  Application components are distributed across locations. Each individual component can have multiple instances located only in a single location.
- **Horizontal hybrid applications**
  Application components are distributed across locations. Each individual component can have multiple instances spanning multiple locations.

  Some components can be aware of their location while others do not have any knowledge of their location and placement. This virtuousness can be achieved with an abstraction layer. This layer, with a modern application framework like microservices, can define how the application is serviced by the placement of application components operating on nodes across clouds.

Placement checklist

Verify required locations. Make sure the application or any of its components are required to operate in, or require certification for, a specific cloud. This can include sovereignty requirements from your company or dictated by law. Also, determine if any on-premises operations are required for a particular location or locale.

Ascertain connectivity dependencies. Required locations and other factors can dictate the connectivity dependencies among your components. As you place the components, determine the optimal connectivity and security for communication among them. Choices include VPN, ExpressRoute, and Hybrid Connections.

Evaluate platform capabilities. For each application component, see if the required resource provider for the application component is available on the cloud and if the bandwidth can accommodate the expected throughput and latency requirements.

Plan for portability. Use modern application frameworks, like containers or microservices, to plan for moving operations and to prevent service dependencies.

Determine data sovereignty requirements. Hybrid applications are geared for accommodating data isolation, such as on a local datacenter. Review the placement of your resources to optimize the success for accommodating this requirement.

Plan for latency. Inter-cloud operations can introduce physical distance between the application components. Ascertain the requirements to accommodate any latency.

Control traffic flows. Handle peak usage and the appropriate and secured communications for personal identifiable information (PII) data when accessed by the front-end in a public cloud.

Scalability

Scalability is the ability of a system to handle increased load on an application, which can vary over time as other factors, and forces, affect the audience size in addition to the size and scope of the application.

For the core discussion of this pillar, see Scalability in Pillars of software quality.

A horizontal scaling approach for hybrid applications allows for adding more instances to meet demand and then disabling them during quieter periods.

In hybrid scenarios, scaling out individual components requires additional consideration when components are spread across clouds. Scaling one part of the application can require the scaling of another. For example, if the number of client connections increases but the application's web services are not scaled out appropriately, the load on the database might saturate the application.

Some application components can scale out linearly, while others have scaling dependencies and might be limited to what extend they are able to scale. For example, a VPN tunnel providing hybrid connectivity for the application components locations has a limit to the bandwidth and latency it can be scaled to. How are components of the application scaled to ensure these requirements are met?

Scalability checklist

Ascertain scaling thresholds. To handle the various dependencies in your application, determine the extent to which application components in different clouds can scale independently of each other, while still meeting the requirements to run the application. Hybrid applications often need to scale particular areas in the application to handle a feature as it interacts and affects the rest of the application. For example, exceeding a number of front-end instances may require scaling the back-end.

Define scale schedules. Most applications have busy periods, so you need to aggregate their peak times into schedules to coordinate optimal scaling.

Use a centralized monitoring system. Platform monitoring capabilities can provide autoscaling, but hybrid applications need a centralized monitoring system that aggregates system health and load. A centralized monitoring system can initiate scaling a resource in one location and scaling a depending resource in another location. Additionally, a central monitoring system can track which clouds autoscale resources and which clouds don't.

Leverage autoscaling capabilities (as available). If autoscaling capabilities are part of your architecture, you implement autoscaling by setting thresholds that define when an application component needs to be scaled up, out, down, or in. An example of autoscaling is a client connection that is autoscaled in one cloud to handle increased capacity, but causes other dependencies of the application, spread across different clouds, to also be scaled. The autoscaling capabilities of these dependent components must be ascertained.

If autoscaling is not available, consider implementing scripts and other resources to accommodate manual scaling, triggered by thresholds in the centralized monitoring system.

Determine expected load by location. Hybrid applications that handle client requests might primarily rely on a single location. When the load of client requests exceeds a threshold, additional resources can be added in a different location to distribute the load of inbound requests. Make sure that the client connections can handle the increased loads and also determine any automated procedures for the client connections to handle the load.

Availability

Availability is the time that a system is functional and working. Availability is usually measured as a percentage of uptime. Application errors, infrastructure problems, and system load can all reduce availability.

For the core discussion of this pillar, see Availability in Pillars of software quality.

Availability checklist

Provide redundancy for connectivity. Hybrid applications require connectivity among the clouds that the application is spread across. You have a choice of technologies for hybrid connectivity, so in addition to your primary technology choice, use another technology to provide redundancy with automated failover capabilities should the primary technology fail.

Classify fault domains. Fault-tolerant applications require multiple fault domains. Fault domains help isolate the point of failure, such as if a single hard disk fails on premises, if a top-of-rack switch goes down, or if the full datacenter is unavailable. In a hybrid application, a location can be classified as a fault domain. With more availability requirements, the more you need to evaluate how a single fault domain should be classified.

Classify upgrade domains. Upgrade domains are used to ensure that instances of application components are available, while other instances of the same component are being serviced with updates or feature upgrades. As with fault domains, upgrade domains can be classified by their placement across locations. You must determine if an application component can accommodate getting upgraded in one location before it is upgraded in another location, or if other domain configurations are required. A single location itself can have multiple upgrade domains.

Track instances and availability. Highly available application components can be available through load balancing and synchronous data replication. You must determine how many instances can be offline before the service is interrupted.

Implement self-healing. In the event an issue causes an interruption to the application availability, a detection by a monitoring system could initiate self-healing activities to the application, such as draining the failed instance and redeploying it. Most likely this requires a central monitoring solution, integrated with a hybrid Continuous Integration and Continuous Delivery (CI/CD) pipeline. The application is integrated with a monitoring system to identify issues that could require redeployment of an application component. The monitoring system can also trigger hybrid CI/CD to redeploy the application component and potentially any other dependent components in the same or other locations.

Maintain service-level agreements (SLAs). Availability is critical for any agreements to maintain connectivity to the services and applications that you have with your customers. Each location that your hybrid application relies on might have its own SLA. These different SLAs can affect the overall SLA of your hybrid application.

Resiliency

Resiliency is the ability for a hybrid application and system to recover from failures and continue to function. The goal of resiliency is to return the application to a fully functioning state after a failure occurs. Resiliency strategies include solutions like backup, replication, and disaster recovery.

For the core discussion of this pillar, see Resiliency in Pillars of software quality.

Resiliency checklist

Uncover disaster-recovery dependencies. Disaster recovery in one cloud might require changes to application components in another cloud. If one or multiple components from one cloud are failed over to another location, either within the same cloud or to another cloud, the dependent components need to be made aware of these changes. This also includes the connectivity dependencies. Resiliency requires a fully-tested application recovery plan for each cloud.

Establish recovery flow. An effective recovery flow design has evaluated application components for their ability to accommodate buffers, retries, retrying failed data transfer, and, if necessary, fall back to a different service or workflow. You must determine what backup mechanism to use, what its restore procedure involves, and how often it's tested. You should also determine the frequency for both incremental and full backups.

Test partial recoveries. A partial recovery for part of the application can provide reassurance to users that all is not unavailable. This part of the plan should ensure that a partial restore doesn't have any side effects, such as a backup and restore service that interacts with the application to gracefully shut it down before the backup is made.

Determine disaster-recovery instigators and assign responsibility. A recovery plan should describe who, and what roles, can initiate backup and recovery actions in addition to what can be backed up and restored.

Compare self-healing thresholds with disaster recovery. Determine an application's self-healing capabilities for automatic recovery initiation and the time required for an application's self-healing to be considered a failure or success. Determine the thresholds for each cloud.

Verify availability of resiliency features. Determine the availability of resiliency features and capabilities for each location. If a location does not provide the required capabilities, consider integrating that location into a centralized service that provides the resiliency features.

Determine downtimes. Determine the expected downtime due to maintenance for the application as a whole and as application components.

Document troubleshooting procedures. Define troubleshooting procedures for redeploying resources and application components.

Manageability

The considerations for how you manage your hybrid applications are critical in designing your architecture. A well-managed hybrid application provides an infrastructure as code that enables the integration of consistent application code in a common development pipeline. By implementing consistent system-wide and individual testing of changes to the infrastructure, you can assure an integrated deployment if the changes pass the tests, allowing them to be merged into the source code.

For the core discussion of this pillar, see Management and dev ops in Pillars of software quality.

Manageability checklist

Implement monitoring. Use a centralized monitoring system of application components spread across clouds to provide an aggregated view of their health and performance. This system includes monitoring both the application components and related platform capabilities. Determine the parts of the application that require monitoring.

Coordinate policies. Each location that a hybrid application spans can have its own policy that covers allowed resource types, naming conventions, tags, and other criteria.

Define and use roles. As a database administrator, you need to determine the permissions required for different personas (like an application owner, a database administrator, and an end user) that need to access application resources. These permissions need to be configured on the resources and inside the application. A role-based access control (RBAC) system allows you to set these permissions on the application resources. These access rights are challenging when all resources are deployed in a single cloud but require even more attention when the resources are spread across clouds. Permissions on resources set in one cloud do not apply to resources set in another cloud.

Use CI/CD pipelines. A Continuous Integration and Continuous Development (CI/CD) pipeline can provide a consistent process for authoring and deploying applications that span across clouds, and to provide quality assurance for their infrastructure and application. This pipeline enables the infrastructure and application to be tested on one cloud and deployed on another cloud. The pipeline even allows you to deploy certain components of your hybrid application to one cloud and other components to another cloud, essentially forming the foundation for hybrid application deployment. A CI/CD system is critical for handling the dependencies application components have for each other during installation, such as the web application needing a connection string to the database.

Manage the life cycle. Because resources of a hybrid application can span locations, each single location's life-cycle management capability needs to be aggregated into a single life-cycle management unit. Consider how they are created, updated, and deleted.

Examine troubleshooting strategies. Troubleshooting a hybrid application involves more application components than the same application that is running in a single cloud. Besides the connectivity between the clouds, the application is running on two platforms instead of one. An important task in troubleshooting hybrid applications is to examine the aggregated health and performance monitoring of the application components.

Security

Security is one of the primary considerations for any cloud application, and it becomes even more critical for hybrid cloud applications.

For the core discussion of this pillar, see Security in Pillars of software quality.

Security checklist

Assume breach. If one part of the application is compromised, ensure there are solutions in place to minimize the spread of the breach, not only within the same location but also across locations.

Monitor allowed network access. Determine the network access policies for the application, such as only accessing the application from a specific subnet and only allow the minimum ports and protocols between the components required for the application to function properly.

Employ robust authentication. A robust authentication scheme is critical for the security of your application. Consider using a federated identity provider that provides single sign-on capabilities and employs one or more of the following schemes: username and password sign-on, public and private keys, two-factor or multi-factor authentication, and trusted security groups. Determine the appropriate resources to store sensitive data and other secrets for application authentication in addition to certificate types and their requirements.

Use encryption. Identify which areas of the application use encryption, such as for data storage or client communication and access.

Use secure channels. A secure channel across the clouds is critical for providing security and authentication checks, real-time protection, quarantine, and other services across clouds.

Define and use roles. Implement roles for resource configurations and single-identity access across clouds. Determine the role-based access control (RBAC) requirements for the application and its platform resources.

Audit your system. System monitoring can log and aggregate data from both the application components and the related cloud platform operations.

Summary

This whitepaper provides a checklist of items that are important to consider during the authoring and designing of your hybrid applications. Reviewing these pillars before you deploy your application prevents you from running into these questions in production outages and potentially requiring you to revisit your design.

It can seem like a time-consuming task beforehand, but you easily get your return on investment if you design your application based on these pillars.

Learn more

For more information, see the following resources:

- Hybrid cloud
- Hybrid cloud applications
- Develop Azure Resource Manager templates for cloud consistency