Microsoft Azure

A key benefit of Azure is creating highly scalable applications using Cloud Services.

Applications can shrink and stretch to accommodate changes in usage, removing the need for expensive on-premises hardware.

A key strategy is to design in scale units, which are a base configuration of web and worker role instances with supporting services such as data stores and caching.

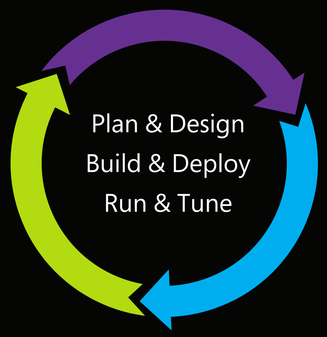
Three reasons to create Azure scalable applications:

DEMAND PEAKS
Your app reaches thousands of users (or more) although usage varies, sometimes greatly.

DISTRIBUTED USERS AND DEVICES
Your users are spread out, even around the globe.

PARTITIONABLE WORKLOADS
Your processes are divided into optimal-size loads of work, since cloud applications scale by adding capacity in chunks.

Note: Not all of these need to be present in your application, however, one that does not exhibit any of these characteristics is probably not an ideal fit.



Microsoft

PLAN AND DESIGN

A highly scalable application requires the use of specific patterns and practices. Designing for optimal performance and scale-out is key. Use the patterns below to help you architect your solution and continually refine your application.

SCALE OUT WITH SCALE UNITS

Use more instances, not bigger hardware. Scale in and out using scale units that are easily duplicated and deployed. Scale units consist of a number of role instances and their support services.

For example, a scale unit could be 3 web roles, 2 worker roles, 1 queue, and 2 SQL Database instances.

SAVING STATE

The durability of a web and worker role instance is not assured, therefore its state (customer data, stage in a workflow, etc.) must be saved externally.

Save state to durable storage (tables, SQL Database, blobs), where other instances can resume the work.

DECOUPLED COMMUNICATIONS

Avoid tying up valuable resources by using an asynchronous decoupled programming method. Web role instances put autonomous messages into a queue for pickup by worker role instances, which continue the work. Throughput is controlled by the number of role instances producing and processing messages. Explore using Azure Service Bus or Storage Queues.

CHUNKY, NOT CHATTY

Network calls require overhead for packet framing, serialization, processing, and so on. Rather than use "chatty" messages, batch them into fewer "chunky" packages. Note, however, that batching can increase latency and exposure to potential data loss.

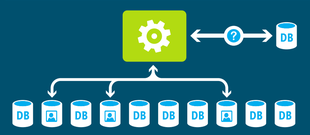
RETRY FOR FAFULT TOLERANCE

Transient errors and throttling are unavoidable in large-scale systems. Instead of simply failing the operation, implement a robust retry strategy across the application to provide resiliency against failures. Too many retries too quickly can add additional load, so also employ a "backoff" strategy that allows the resource to recover by waiting after multiple retries.

CACHING

Caching improves performance by storing recently used data for immediate reuse. Application throughput and latency are typically bound by how quickly data and context can be retrieved, shared, and updated. **Microsoft Azure Cache** provides caching as a service.

FAN-OUT QUERIES

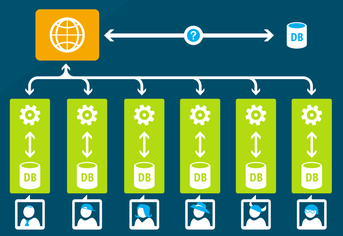


Database lookup logic is placed in a cloud service. To find data, that cloud service determines the databases to query. The query is then fanned out to those databases.

HORIZONTAL PARTITIONING

As user data increases, the need for storage increases. The database must be partitioned. This graphic shows a horizontal partition (also known as a *shard*) where intact tables are separated into individual databases. Each user's data can be distributed to particular databases. You can create and delete databases very quickly.

VERTICAL AFFINITY



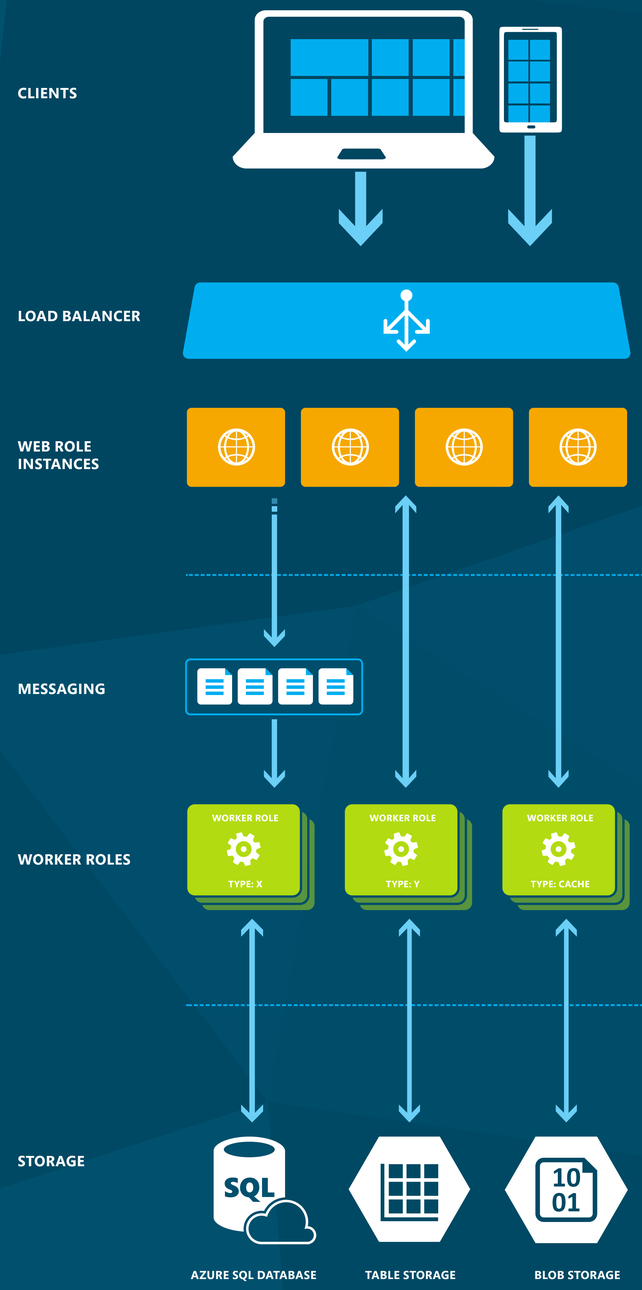
When many users access data simultaneously, traffic becomes a problem as scale increases. Design your processes to access exclusive partitions to minimize traffic and resource usage.

For example, assume databases are partitioned by user. Ideally all operations that access a single user's data are routed to a specific set of service instances. Those instances access a single database partition holding all the user's data.

BUILD AND DEPLOY

Applications that are built on Cloud Services are easily scaled. Web and worker instances can be increased and decreased at will. Workloads can be distributed using messaging, such as queues or Service Bus Topics.

Tables and blobs provide massive storage capacity and SQL Database supplies relational capabilities. Other services such as caching can be easily integrated into a service.



RUN AND TUNE

This phase contains the processes that refine the application, keep it running, and enable scaling out (and in) as needed. Tuning your application takes time and requires instrumentation and monitoring.

It's a good practice to continually assess the metrics and balance against running costs.

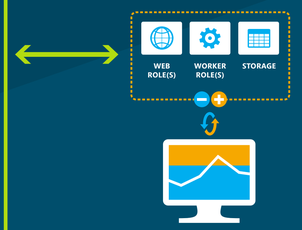
VISIBILITY & MONITORING

Strategically instrument the app to monitor potential bottlenecks. There are two kinds of monitoring:

INTERNAL: Monitoring processes inside the system is essential to determine when additional scale-out is needed.

EXTERNAL: Monitor the performance from outside the application to ensure service performance is within acceptable ranges.

Internal and external monitoring are available through **Application Insights**. **Azure Management Services** also enables you to create rules and alerts for monitoring.



SCALE: BIGGER, BETTER, FASTER

With visibility into the app, you can control scale with more precision. To automate, a separate process monitors the system's vital signs. When a threshold is crossed a new scale unit is deployed. When a lower threshold is crossed, a scale unit can be removed.

Azure Web Sites lets you scale manually, or use its Autoscaling feature to scale up or down as the thresholds you set are crossed.

AUTOMATION: SCRIPT FOR SUCCESS

Maintaining a running, highly scaled application involves repeating operations on a regular basis. Concurrently develop a library of scripts that can be run on multiple deployments when needed. You can manage Azure services with the **Service Management** API. Management libraries are available for a variety of languages, including Node.js, Java, and C#.

LOAD TESTING: GETTING LOADED

Load test the system with both stress tests and by simulating real-life usage. Vary the load size to avoid surprises! Ensure that responsiveness meets user requirements, and that the entire system is resilient. Create load tests with **Visual Studio** that check your system's ability to meet the users' needs. No need to set up virtual machines for load testing, just run the tests with **Visual Studio Online**.

Scaling Applications Using Microsoft Azure Cloud Services

Like it? Get it.
http://gettag.mobi

Created by the Azure Poster Team Email: AzurePoster@microsoft.com