


Design to scale out

08/30/2018 • 2 minutes to read • Contributors 

In this article

[Design your application so that it can scale horizontally](#)

[Recommendations](#)

Design your application so that it can scale horizontally

A primary advantage of the cloud is elastic scaling — the ability to use as much capacity as you need, scaling out as load increases, and scaling in when the extra capacity is not needed. Design your application so that it can scale horizontally, adding or removing new instances as demand requires.

Recommendations

Avoid instance stickiness. Stickiness, or *session affinity*, is when requests from the same client are always routed to the same server. Stickiness limits the application's ability to scale out. For example, traffic from a high-volume user will not be distributed across instances. Causes of stickiness include storing session state in memory, and using machine-specific keys for encryption. Make sure that any instance can handle any request.

Identify bottlenecks. Scaling out isn't a magic fix for every performance issue. For example, if your backend database is the bottleneck, it won't help to add more web servers. Identify and resolve the bottlenecks in the system first, before throwing more instances at the problem. Stateful parts of the system are the most likely cause of bottlenecks.

Decompose workloads by scalability requirements. Applications often consist of multiple workloads, with different requirements for scaling. For example, an application might have a public-facing site and a separate administration site. The public site may experience sudden surges in traffic, while the administration site has a smaller, more predictable load.

Offload resource-intensive tasks. Tasks that require a lot of CPU or I/O resources should be moved to [background jobs](#) when possible, to minimize the load on the front end that is handling user requests.

Use built-in autoscaling features. Many Azure compute services have built-in support for autoscaling. If the application has a predictable, regular workload, scale out on a schedule. For example, scale out during business hours. Otherwise, if the workload is not predictable, use performance metrics such as CPU or request queue length to trigger autoscaling. For autoscaling best practices, see [Autoscaling](#).

Consider aggressive autoscaling for critical workloads. For critical workloads, you want to keep ahead of demand. It's better to add new instances quickly under heavy load to handle the additional traffic, and then gradually scale back.

Design for scale in. Remember that with elastic scale, the application will have periods of scale in, when instances get removed. The application must gracefully handle instances being removed. Here are some ways to handle scale in:

- Listen for shutdown events (when available) and shut down cleanly.
- Clients/consumers of a service should support transient fault handling and retry.
- For long-running tasks, consider breaking up the work, using checkpoints or the [Pipes and Filters](#) pattern.
- Put work items on a queue so that another instance can pick up the work, if an instance is removed in the middle of processing.