Deploying Azure applications for resiliency and availability

04/10/2019 • 4 minutes to read • Contributors

In this article

As you provision and update Azure resources, application code, and configuration settings, a repeatable and predictable process will help you avoid errors and downtime. We recommend automated processes for deployment that you can run on demand and rerun if something fails. After your deployment processes are running smoothly, process documentation can keep them that way.

Automate processes

To activate resources on demand, deploy solutions rapidly, minimize human error, and produce consistent and repeatable results, be sure to automate deployments and updates.

Automate as many processes as possible

The most reliable deployment processes are automated and *idempotent* — that is, repeatable to produce the same results.

- To automate provisioning of Azure resources, you can use Terraform, Ansible, Chef, Puppet, Azure PowerShell, Azure command-line interface (CLI), or Azure Resource Manager templates.
- To configure VMs, you can use cloud-init (for Linux VMs) or Azure Automation State Configuration (DSC).
- To automate application deployment, you can use Azure DevOps Services, Jenkins, or other CI/CD solutions.

As a best practice, create a repository of categorized automation scripts for quick access, documented with explanations of parameters and examples of script use. Keep this documentation in sync with your Azure deployments, and designate a primary person to manage the repository.

Automation scripts can also activate resources on demand for disaster recovery.

Use code to provision and configure infrastructure

This practice, called *infrastructure as code,* may use a declarative approach or an imperative approach (or a combination of both).

- Resource Manager templates are an example of a declarative approach.
- PowerShell scripts are an example of an imperative approach.

Practice immutable infrastructure

In other words, don't modify infrastructure after it's deployed to production. After ad hoc changes have been applied, you might not know exactly what has changed, so it can be difficult to troubleshoot the system.

Automate and test deployment and maintenance tasks

Distributed applications consist of multiple parts that must work together. Deployment should take advantage of proven mechanisms, such as scripts, that can update and validate configuration and automate the deployment process. Test all processes fully to ensure that errors don't cause additional downtime.

Implement deployment security measures

All deployment tools must incorporate security restrictions to protect the deployed application. Define and enforce deployment policies carefully, and minimize the need for human intervention.

Deploy to multiple regions and instances

An application that depends on a single instance of a service creates a single point of failure. To improve resiliency and scalability, provision multiple instances.

- For Azure App Service, select an App Service plan that offers multiple instances.
- For Azure Cloud Services, configure each of your roles to use multiple instances.
- For Azure Virtual Machines, ensure that your architecture includes more than one VM and that each VM is included in an availability set.

Consider deploying across multiple regions

We recommend deploying all but the least critical applications and application services across multiple regions. If your application is deployed to a single region, in the rare event that the entire region becomes unavailable, the application will also be unavailable.

If you choose to deploy to a single region, consider preparing to redeploy to a secondary region as a response to an unexpected failure.

Redeploy to a secondary region

If you run applications and databases in a single, primary region with no replication, your recovery strategy might be to redeploy to another region. This solution is affordable but most appropriate for non-critical applications that can tolerate longer recovery times.

If you choose this strategy, automate the redeployment process as much as possible and include redeployment scenarios in your disaster response testing.

To automate your redeployment process, consider using Azure Site Recovery.

Use staging and production environments

With good use of staging and production environments, you can push updates to the production environment in a highly controlled way and minimize disruption from unanticipated deployment issues.

- *Blue-green deployment* involves deploying an update into a production environment that's separate from the live application. After you validate the deployment, switch the traffic routing to the updated version. One way to do this is to use the staging slots available in Azure App Service to stage a deployment before moving it to production.

- *Canary releases* are similar to blue-green deployments. Instead of switching all traffic to the updated application, you route only a small portion of the traffic to the new deployment. If there's a problem, revert to the old deployment. If not, gradually route more traffic to the new version. If you're using Azure App Service, you can use the Testing in production feature to manage a canary release.

Create a rollback plan for deployment and updates

If a deployment fails, your application could become unavailable. To minimize downtime, design a rollback process to go back to a last-known good version. Include a strategy to roll back changes to databases and any other services your app depends on.

If you're using Azure App Service, you can set up a last-known good site slot and use it to roll back from a web or API app deployment.

Log and audit deployments

To capture as much version-specific information as possible, implement a robust logging strategy. If you use staged deployment techniques, more than one version of your application will be running in production. If a problem occurs, determine which version is causing it.

Document release processes

Without detailed release process documentation, an operator might deploy a bad update or might improperly configure settings for your application. Clearly define and document your release process, and ensure that it's available to the entire operations team.

Next steps

Monitor application health for reliability