Engineer

Bicep

Azure's own language for declaring infrastructure

What it is

Bicep is a language from Microsoft for writing Azure infrastructure down as text: which resources exist, what they are called, how each is configured and how they connect to one another. A Bicep file lists no steps. It states the end result, and Azure Resource Manager, the layer Azure management requests pass through, works out what to create, change or leave alone to get there. Someone asking “what is Bicep?” usually wants to know whether it replaces clicking through the Azure portal. It does, and that is most of its value.

Underneath, Bicep is a friendlier way to write something Azure already understood. Resource Manager has long accepted deployment templates written in JSON, which are complete but hard to read and harder to review. During a deployment the Bicep command-line tool converts a `.bicep` file into one of those JSON templates, and Resource Manager deploys it exactly as it would a hand-written one. Microsoft states that Bicep supports every resource type and API version as soon as a resource provider introduces it, previews included, so a new service is usable without waiting for anyone to update a plug-in. Existing JSON templates can be decompiled into Bicep, which makes moving an older estate across a mechanical job rather than a rewrite.

A small file shows the shape. Parameters come first, then resources, each with a symbolic name the rest of the file can refer to:

```
param env string
param location string = resourceGroup().location

resource store 'Microsoft.Storage/storageAccounts@2023-05-01' = {
  name: 'st${env}${uniqueString(resourceGroup().id)}'
  location: location
  sku: {
```

```
    name: 'Standard_LRS'
  }
  kind: 'StorageV2'
}
```

When another resource mentions `store.id`, Bicep infers that it has to wait for the storage account, so nobody writes the order by hand, and Resource Manager deploys whatever does not depend on anything else in parallel. A file can call another file as a module, which lets a team describe “a database with its network rules” once and reuse it everywhere. Values that differ per environment live in a separate `.bicepparam` file. Microsoft is explicit that Bicep is free, open source and not meant as a general programming language: it has loops, conditions and functions, but only in service of describing resources.

How does Bicep compare with the alternatives? Against ARM JSON, it is the same engine with a far shorter source. Against Terraform, the main differences are reach and state: Terraform covers many clouds and keeps a state file of its own, while Bicep’s Azure Resource Manager workflow keeps no state file of its own, because Azure stores the deployment state. Bicep extensions also support resources outside Resource Manager, including Kubernetes and Microsoft Graph. Against Pulumi, which describes infrastructure in general-purpose languages such as TypeScript or Go, Bicep trades flexibility for a smaller language that is quick to read in review.

Use the concept in a decision

Put “What is Bicep?” to work in a real decision. We can help relate the concept to the system you are building or buying. The whole setup written down as files, so it can be recreated exactly and never lives only in one person's head.

Explore our services Get in touch →

Test the assumptions behind the term

Use the explanation and linked work to frame questions for your project. We can examine the assumptions with you before a tool is chosen. Wrote 1,000 RBAC rules to secure Linux servers — 1,000 access rules written to close a security gap

Wrote 1,000 RBAC rules to secure Linux servers Get in touch →

Why it matters

Infrastructure built by hand has no history. A setting changed in the portal leaves no reason behind it, nobody reviews it before it happens, and rebuilding the environment after a bad day depends on someone remembering every click. Once the infrastructure is a set of files in the repository, it gets what application code already has: a pull request, a reviewer, a history and a way back.

The second gain is sameness. Development, staging and production built from the same files with different parameters differ only where the parameters say they do. Bicep deployments are idempotent, which in Microsoft's words means the same file deployed many times gives "the same resource types in the same state". A deployment is therefore safe to repeat, and a fresh environment is the same command pointed at a new resource group.

The third is visibility before change. The what-if operation compares a file with what is running and lists each resource it would create, modify or delete before anything is touched. Paired with a pull request, that preview turns "I think this only changes the firewall rule" into something a reviewer can check.

The costs are worth naming. Resource coverage depends on the deployment workflow and its extensions, so DNS at another provider or resources in a second cloud may still need a separate tool. In the Resource Manager workflow, Bicep describes Azure resources; packages and files inside a virtual machine belong to configuration management such as Ansible. And Resource Manager's defaults around deletion and redeployment surprise people who expect a desired-state tool to remove what the file no longer mentions, which section four covers.

What it looks like in practice

Several of the Azure projects on this site rest on Bicep, and three show what it does for a team over time.

On one platform the whole Azure estate is written in Bicep. Every environment comes out of the same templates: the api and www as Azure Container Apps, a PostgreSQL Flexible Server, Redis for caching, Front Door with a WAF policy in front, and the network underneath it (VNet, NSG, private DNS), with Log Analytics wired in for diagnostics and images pulled from the project's Azure Container Registry. Because it is all parameterised, standing up a new environment or changing an existing one is a pull request. The page reports the result as environments that became reproducible and reviewable, and drift that stopped being a mystery because the code is the source of truth.

The same templates carry the release path. Releases move from development to staging to production, and the Bicep keeps the target infrastructure identical from one environment to the next, so a build is never promoted into a subtly different place. Configuration that differs per environment is kept apart from the secrets, which lets one built artefact pass through every stage without a rebuild.

And Bicep holds part of the recovery plan. Backup retention is configured in the Bicep beside the database it protects, so point-in-time restore is a property of the template rather than a setting someone once clicked. The environment around the database is code too, which matters because restoring data into an environment that has to be rebuilt from memory is hardly recovery. The page then names what remains open: geo-redundancy is switched off, and the recovery-time objective is proposed rather than measured. A template shows what is configured; only a drill shows how long a restore takes.

Where it goes wrong

Most Bicep trouble comes from expecting it to behave like a tool that owns everything in a resource group. By default it owns only what the file mentions, and only in the way the file says.

A setting changed in the portal disappears at the next deployment. Redeploying a resource reapplies all of its properties, and any property the template leaves out is reset to its default. A hotfix made by hand during an incident is undone by the next routine release. Treat the file as the full final state of each resource, write every non-default value into it, and make urgent changes through a pull request. A deployment stack's deny settings can block edits outside the deployment altogether.

A resource removed from the template keeps running and keeps costing money. The default deployment mode is incremental: resources in the resource group that the file does not mention are left alone. Delete a database from the code and it stays in Azure. Microsoft's answer is deployment stacks. A stack records the resources it manages, and its `actionOnUnmanage` setting decides whether one that drops out of the file is detached or deleted. Complete mode also deletes what the file omits, but Microsoft recommends against it and is phasing it out.

The what-if preview lists changes nobody made. A property the file leaves out, but that Azure fills in with a default during deployment, can be reported as deleted, and Microsoft calls this noise. Any property set from the `reference` function is reported as changing every time, and values such as `utcNow()` or a secure parameter exist only during the deployment, so what-if shows them unevaluated. Run it on every pull request, read it as a draft, and set the noisy properties explicitly so the real changes stand out:

```
az deployment group what-if \
  --resource-group rg-staging \
  --template-file main.bicep \
  --parameters staging.bicepparam
```

A password shows up in the deployment history. A plain string parameter is recorded with the deployment, where anyone with read access to the deployment history can see it. Mark secret parameters with the `@secure()` decorator, which keeps the value out of deployment history and logs, and keep the secret itself in Key Vault rather than in a parameter file in the repository.

Further reading

- What is Bicep? — Microsoft's own overview: the language, its benefits and what it is not trying to be.
- Bicep modules — splitting a deployment into files that call each other.
- Parameters in Bicep files — types, decorators, defaults, and what `@secure()` keeps out of the history.
- Deployment modes — incremental against complete, and what each does with resources the file does not mention.
- Bicep deployment what-if operation — previewing a deployment, the change types and the known sources of noise.
- Deployment stacks — managing resources as one unit, with deletion and deny settings.

Glossary

Choose for your actual constraints

Bring the deployment path, the operational risks and the people on call. We can scope a change with checks, a recovery route and clear ownership.

See what it costs Get in touch →

Make the terminology practical

Ask how “What is Bicep?” applies to your organisation. A concrete example from your current system is enough to start the discussion.

Explore our services

welcome@engineer.company +45 50 10 61 84 Copenhagen, Denmark

Take this page with you

Contact card

Carefully Crafted With Love by D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

Copyright © 2025 – present · Engineer ApS · All rights reserved.

Printed from https://engineer.company/glossary/bicep/