Engineer

Bicep

Azures eget språk för att deklarera infrastruktur

Vad det är

Bicep är ett språk från Microsoft för att skriva ned infrastruktur i Azure som text: vilka resurser som finns, vad de heter, hur var och en är konfigurerad och hur de hänger ihop med varandra. En Bicep-fil räknar inte upp några steg. Den anger slutresultatet, och Azure Resource Manager (ARM), det lager som hanteringsbegäranden till Azure passerar, räknar själv ut vad som ska skapas, ändras eller lämnas orört för att komma dit. Den som frågar ”vad är Bicep?” vill oftast veta om det ersätter att klicka sig igenom Azure-portalen. Det gör det, och det är det mesta av dess värde.

Under ytan är Bicep ett vänligare sätt att skriva något som Azure redan förstod. Resource Manager har länge tagit emot mallar för driftsättning skrivna i JSON, som är fullständiga men svåra att läsa och ännu svårare att granska. Under en driftsättning omvandlar Bicep-verktyget för kommandoraden en `.bicep` -fil till en sådan JSON-mall, och Resource Manager driftsätter den precis som en mall skriven för hand. Microsoft uppger att Bicep stöder varje resurstyp och API-version så snart en resursprovider inför den, förhandsversioner inräknade, så en ny tjänst går att använda utan att vänta på att någon uppdaterar ett tillägg. Befintliga JSON-mallar kan dekompileras till Bicep, vilket gör flytten av en äldre uppsättning till ett mekaniskt arbete snarare än en omskrivning.

En liten fil visar formen. Parametrarna kommer först och resurserna sedan, var och en med ett symboliskt namn som resten av filen kan hänvisa till:

```
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'
}
```

När en annan resurs nämner `store.id` drar Bicep slutsatsen att den måste vänta på lagringskontot, så ordningen behöver aldrig skrivas för hand, och Resource Manager driftsätter parallellt allt som inte beror på något annat. En fil kan anropa en annan fil som en modul, vilket låter ett team beskriva ”en databas med sina nätverksregler” en gång och återanvända den överallt. Värden som skiljer sig mellan miljöerna ligger i en separat `.bicepparam` -fil. Microsoft skriver uttryckligen att Bicep är gratis, har öppen källkod och inte är tänkt som ett allmänt programspråk: det har loopar, villkor och funktioner, men bara för att beskriva resurser.

Hur står sig Bicep mot alternativen? Mot ARM-mallar i JSON är det samma motor med en mycket kortare källtext. Mot Terraform ligger de stora skillnaderna i räckvidd och tillstånd: Terraform täcker många moln och håller en egen tillståndsfil, medan Biceps arbetsflöde med Azure Resource Manager inte för någon egen tillståndsfil, eftersom Azure lagrar driftsättningens tillstånd. Biceps tillägg stöder även resurser utanför Resource Manager, bland annat Kubernetes och Microsoft Graph. Mot Pulumi, som beskriver infrastruktur i allmänna programspråk som TypeScript eller Go, byter Bicep bort flexibilitet mot ett mindre språk som går fort att läsa vid en granskning.

Använd begreppet i ett beslut

Använd ”Vad är Bicep?” i ett konkret beslut. Vi kan knyta begreppet till systemet ni bygger eller köper. Hela uppsättningen nedskriven som filer, så att den kan återskapas exakt och aldrig bara finns i en enda persons huvud.

Se våra tjänster Hör av er →

Granska antagandena bakom ordet

Använd förklaringen och det länkade arbetet för att formulera frågor till projektet. Vi kan undersöka antagandena tillsammans innan ett verktyg väljs. Skrev 1 000 RBAC-regler som säkrar Linux-servrar — 1,000 åtkomstregler, skrivna för att täppa till en säkerhetslucka

Skrev 1 000 RBAC-regler som säkrar Linux-servrar Hör av er →

Varför det spelar roll

Infrastruktur som byggs för hand har ingen historik. En inställning som ändras i portalen lämnar inget skäl efter sig, den granskas inte innan den görs, och att bygga upp miljön igen efter en dålig dag hänger på att någon minns varje klick.

När infrastrukturen väl är en uppsättning filer i repot får den det som applikationskoden redan har: en pull request, en granskare, en historik och en väg tillbaka.

Den andra vinsten är likformighet. Utveckling, staging och produktion som byggs ur samma filer med olika parametrar skiljer sig bara där parametrarna säger att de ska göra det. Driftsättningar med Bicep är idempotenta, vilket med Microsofts ord innebär att samma fil, driftsatt många gånger, ger samma resurstyper i samma tillstånd. En driftsättning är därför säker att upprepa, och en ny miljö är samma kommando riktat mot en ny resursgrupp.

Den tredje är insyn före ändringen. Operationen what-if jämför en fil med det som körs och listar varje resurs som den skulle skapa, ändra eller ta bort, innan något rörs. Tillsammans med en pull request gör den förhandsvisningen ”jag tror att det här bara ändrar en regel i brandväggen” till något som en granskare kan kontrollera.

Kostnaderna är värda att nämna. Vilka resurser som stöds beror på driftsättningsflödet och dess tillägg, så DNS hos en annan leverantör eller resurser i ett andra moln kan fortfarande behöva ett separat verktyg. I arbetsflödet med Resource Manager beskriver Bicep Azure-resurser; paket och filer inne i en virtuell maskin hör till konfigurationshantering som Ansible. Och Resource Managers standardbeteende kring borttagning och ny driftsättning överraskar den som väntar sig att ett verktyg för önskat tillstånd ska ta bort det som filen inte längre nämner, vilket det fjärde avsnittet tar upp.

Så ser det ut i praktiken

Flera av Azure-projekten på den här webbplatsen vilar på Bicep, och tre av dem visar vad språket gör för ett team över tid.

På en plattform, i rollen som DevOps-ingenjör, är hela Azure-uppsättningen skriven i Bicep, och varje miljö byggs ur samma mallar. Mallarna beskriver api och www som Azure Container Apps i en hanterad miljö, en PostgreSQL Flexible Server, Redis för cachning och Front Door med en WAF-policy längst fram. Under dem ligger nätverket, med VNet, NSG och privat DNS, och Log Analytics är inkopplat för diagnostik medan containeravbildningarna hämtas från projektets Azure Container Registry. Allt är parametriserat, så att sätta upp en ny miljö eller

ändra en befintlig är en pull request. Sidan redovisar resultatet som miljöer som gick att återskapa och granska, och avvikelser som slutade vara ett mysterium eftersom sanningskällan är koden.

Samma mallar bär vägen för releaser. I rollen som Site Reliability Engineer går releaserna från utveckling till staging och sedan till produktion, och Bicep håller målinfrastrukturen identisk från en miljö till nästa, så ett bygge förs aldrig vidare till en subtilt annorlunda plats. Hemligheterna hålls isär från den konfiguration som varierar mellan utveckling, staging och produktion, vilket låter en och samma byggda artefakt passera varje steg utan att byggas om.

Bicep bär också en del av återställningsplanen. Hur länge säkerhetskopiorna sparas ställs in i Bicep, bredvid databasen som de skyddar, så återställning till valfri tidpunkt är en egenskap hos mallen och inte en inställning som någon en gång klickade fram. Miljön runt databasen är också kod, och det spelar roll, eftersom data som återställs till en miljö som måste byggas upp igen ur minnet knappast räknas som en återställning. Sidan namnger sedan det som fortfarande är öppet: georedundans är avslagen, och målet för återställningstid är föreslaget snarare än uppmätt. En mall visar vad som är konfigurerat; bara en övning visar hur lång tid en återställning tar.

Var det går fel

Det mesta besväret med Bicep kommer av att man väntar sig att det ska bete sig som ett verktyg som äger allt i en resursgrupp. Som standard äger det bara det som filen nämner, och bara på det sätt som filen anger.

En inställning som ändrats i portalen försvinner vid nästa driftsättning. När en resurs driftsätts på nytt tillämpas alla dess egenskaper igen, och varje egenskap som mallen utelämnar återställs till sitt standardvärde. En snabbrättning som görs för hand under en incident görs ogjord av nästa rutinmässiga release. Behandla filen som det fullständiga sluttillståndet för varje resurs, skriv in varje värde som avviker från standard, och gör brådskande ändringar via en pull request. Med nekandeinställningarna i en deployment stack går det att blockera ändringar utanför driftsättningen helt och hållet.

En resurs som tagits bort ur mallen fortsätter att köras och att kosta pengar. Standardläget för driftsättning är inkrementellt: resurser i resursgruppen som filen inte nämner lämnas i fred. Ta bort en databas ur koden, och den ligger ändå kvar i Azure. Microsofts svar är deployment stacks. En stack registrerar de

resurser som den hanterar, och dess inställning `actionOnUnmanage` avgör om en resurs som faller ur filen kopplas loss eller tas bort. Fullständigt läge (complete) tar också bort det som filen utelämnar, men Microsoft avråder från det och håller på att fasa ut det.

Förhandsvisningen what-if listar ändringar som ingen har gjort. En egenskap som filen utelämnar, men som Azure fyller i med ett standardvärde under driftsättningen, kan rapporteras som borttagen, och Microsoft kallar det brus. Varje egenskap som sätts med funktionen `reference` rapporteras som ändrad varje gång, och värden som `utcNow()` eller en säker parameter finns bara under driftsättningen, så what-if visar dem outvärderade. Kör den på varje pull request, läs den som ett utkast, och sätt de brusiga egenskaperna uttryckligen så att de verkliga ändringarna syns:

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

Ett lösenord dyker upp i driftsättningshistoriken. En vanlig strängparameter registreras tillsammans med driftsättningen, där alla med läsrätt till historiken kan se den. Märk hemliga parametrar med dekoratorn `@secure()`, som håller värdet borta från driftsättningshistorik och loggar, och förvara själva hemligheten i Key Vault i stället för i en parameterfil i repot.

Läs mer

- What is Bicep? — Microsofts egen översikt: språket, dess fördelar och vad det inte försöker vara.
- Bicep modules — att dela upp en driftsättning i filer som anropar varandra.
- Parameters in Bicep files — typer, dekoratorer, standardvärden och vad `@secure()` håller borta från historiken.
- Deployment modes — inkrementellt mot fullständigt läge, och vad vart och ett gör med resurser som filen inte nämner.
- Bicep deployment what-if operation — att förhandsvisa en driftsättning, ändringstyperna och de kända källorna till brus.
- Deployment stacks — att hantera resurser som en enhet, med inställningar för borttagning och nekande.

Ordlista

Välj utifrån era faktiska förutsättningar

Ta med driftsättningsvägen, driftsriskerna och dem som har jour. Vi kan avgränsa en förändring med kontroller, en väg tillbaka och tydligt ansvar.

Se vad det kostar Hör av er →

Gör begreppen praktiska

Fråga hur ”Vad är Bicep?” gäller i er organisation. Ett konkret exempel från ert nuvarande system räcker för att börja samtalet.

Se våra tjänster

welcome@engineer.company +45 50 10 61 84 Köpenhamn, Danmark

Ta med dig den här sidan

Visitkort

Omsorgsfullt skapad med kärlek av D.

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

Upphovsrätt © 2025 – nu · Engineer ApS · Alla rättigheter förbehållna.

Utskrivet från https://engineer.company/sv/glossary/bicep/