* No SSH, RDP into a web app
* No real difference between Web App, API App, Mobile App
  + Azure Functions can be run as App Services
* Only a handful of languages are supported to be run as App Service
* App Insights is only for windows based app service (not Linux)
* PaaS
* Web Apps + API Apps + Mobile Apps + Function Apps
* WebJobs: background service
* **Hybrid Data Connections**: connect to onpremises data
* **Virtual Network Endpoints:** Connect to VMs running Azure as a service
* Restrict access to IP addresses
* Custom domains
  + SSL
* Autoscaling is available
* When stopping App Service, the cost is still being calculate (it just stops access, no cost saving)



Mobile Apps:

* Azure Notification Hubs integration to push notifications for clients
* If using NodeJS: Easy tables (backed by Azure Sql)

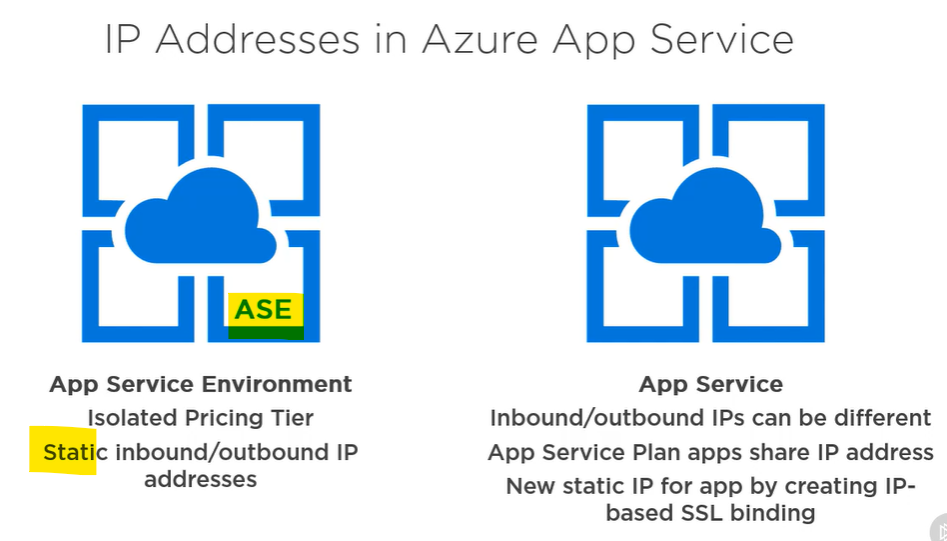
App service plan

* Container for app services
* Defines set of compute resources to use (~serverfarm)
* Region
* Number of VMs (scaleout)
* Size of VM instances (CPU, RAM, storage)
* Pricing tier



From STANDARD TIER: we get out own dedicated VM, unshared with other customers

**App Service Environment:** ASE, compute isolation (dedicated infrastructure) 🡪 ISOLATED TIER

* Windows/linux web/api apps
* Docker cotainers
* Mobile apps
* Function apps
* + Network isolation
  + Not be accessible from internet, at all
  + ExpressRoute, from on-premise network only  
    

**App Service on Linux:** docker images as WEB APPS

* nodeJs, Java, PHP, NET Core, Ruby, Go, Apache Tomcat + custom images
* no VM integration
* Basic, Standard, Premium, Isolated tiers
* CI
* Deployment slots
* Remote debugging
* Logging
* App insights
* Authentication
* Auto scale
* Auto heal (restart)

API/WEB APPS:

* Formerly azure websites
* No difference between the two
* Can be used as microservices
  + Put Azure API Management infront of them though

MOBILE APPS:

* Formerly mobile services
* Same as api/web apps, but also does:
  + EasyTables, EasyAPIs (deploy these without backend) 🡪 only for NodeJS
    - Source code created in browser
    - No CI
    - EasyTables: exposed as OData endpoints, backed by Azure SQL

FUNCTION APPS:

* Small code to run without host application
* Event driven model: direct calls, listens to events
* Serverless
  + Consumption based plan: pay for execution time and resources used only
  + If they are running frequently, use App Service Plan
* Triggers
  + HTTP request
  + timer event
  + storage account events,
  + CosmosDB events
  + Azure Event Hubs, Grid
  + WebHook events
* Supported languages: C#, JavaScript, F#, Java
* They kinda go to sleep after a while, **but there is an ‘Always On’ setting to mitigate this**

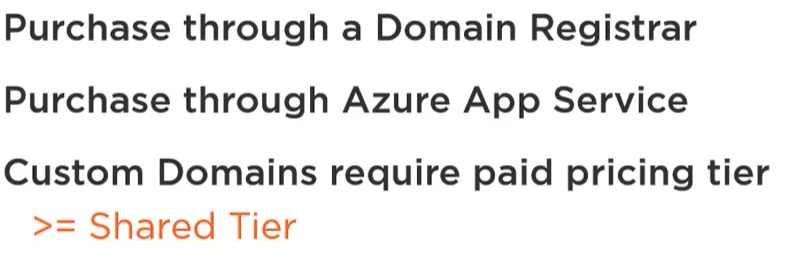
WEB JOBS:

* Run along other, existing App Services
  + Run program/script in the background
    - Exe, bat, cmd (Windows commands)
    - Powersheell (ps1)
    - Bash (sh)
    - PHP
    - Python
    - NodeJs
    - Java (jar)
  + Execution models:
    - Run continuously
    - Run on schedule
    - Run when triggered
  + WebJobs SDK for custom development (Azure Functions are built on top of this)
    - More control with webjobs, than Functional Apps
  + NO ADDITIONAL COST, BECAUSE YOU ALREADY PAY FOR THE APP SERVICE THEY RUN WITHIN
  + PaaS restrictions: no write to registry, no access to event logs, **GDI+ not allowed** (image comparioson, PDF comparison, etc)

Deploy:

* Straight from visual studio
  + Import publish profile, if we don’t have much roles (contriobutor is necessary)
  + Publish dialog, connect to Azure
* CI
  + BitBucket
  + GitHub
  + DevOps
* Local git repo
* OneDrive/Dropbox deployment
* FTP
* Zip/War
  + We can do this with Kudu> Tools > Zip publish deploy

# Custom domains



(Azure uses GoDaddy)

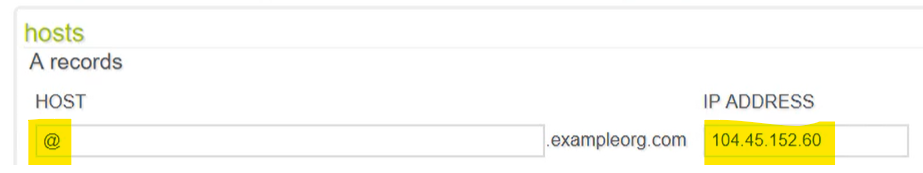
**Azure DNS Service:** delegate DNS managing to Azure portal (eg. We used Google to register a URL, and there specified the DNS endpoint to Azure DNS Service) 🡪 middle man, but now everything’s available at one place, the Azure Portal, plus you get:

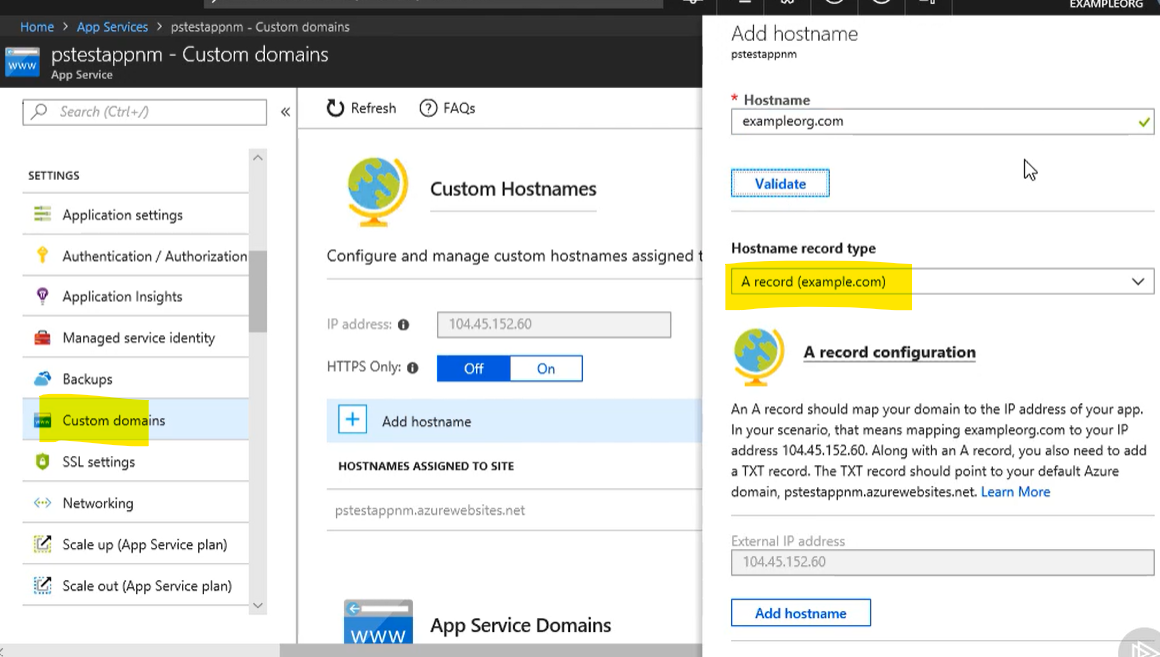
* RBAC (role based access control)
* Activity Logs
* Purchase custom domains in App Service (automatically managed by Azure DNS)

Root domain (apex): example.org (usually @ identifies the root domain)  
Subdomain: test.example.org (or even **www**.example.org)  
DNS server:

* **A Record**: maps domain to an IP address
* **CNAME Record**: maps subdomain to another domain/subdomain (not supposed to use with root domain, only for subdomains 🡪 might conflict with other record types)
* **TXT Record**: stored strings for any purposes

Root domain point to App Service:

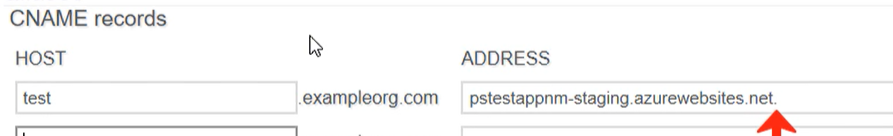
* A records:  
    
  @ means root, the IP is from the Azure App service
  + Now Azure needs a TXT record, to verify that indeed, I own the domain
  +   
    can be deleted after configuration

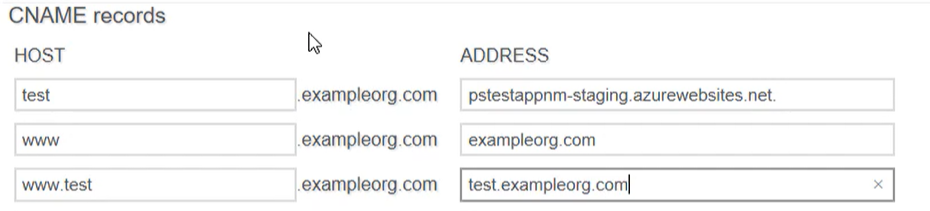


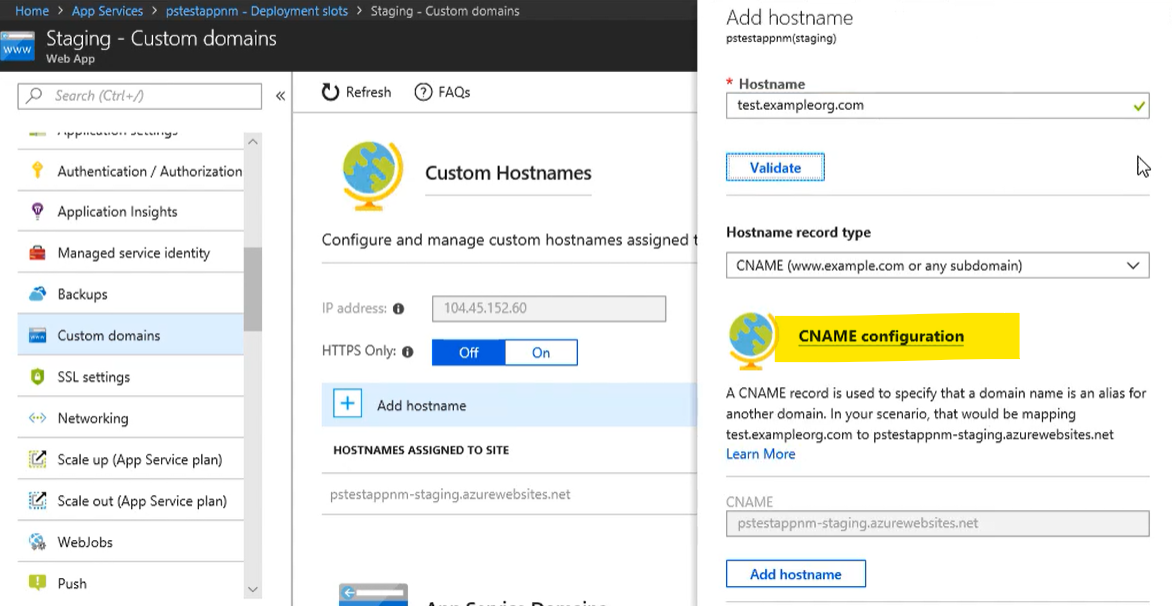
NOTE: HTTPS WONT WORK AT THIS POINT YET

Add subdomains:

Custom DNS provider:

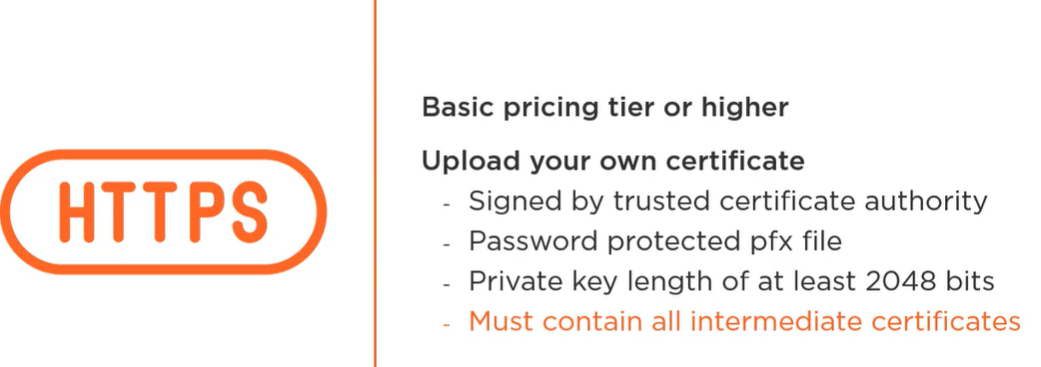
  
trailing dot means absolute url, shouldn’t be suffixed





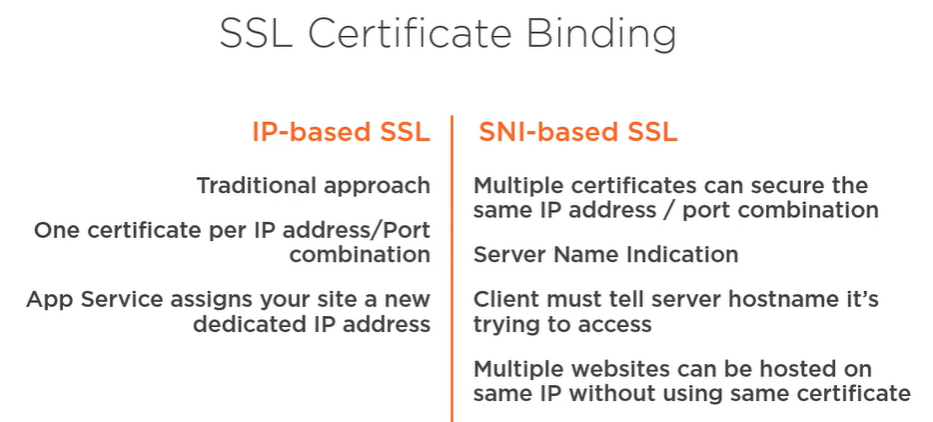
(do the same with www. Tags)

Note: when purchasing domains from Azure directly, privacy is included for free (its usually a paid option at other providers)



App Service Certificate

* Create and manage certis in Azure
* Store in Azure Key Vault
* Uses GoDaddy again



With SSL certis:

* Subject name must be the (sub)domain name
* Or Subject Alternative Name must contain domain name (it usually contains www already)
* OR JUST BUY WILDCARD CERTIFICATE
  + Matches all subdomains of root domain
  + Flexible, but more expensive