به نام خدا

Azure pipelines

**1 . پایپ لاین چیست ؟**

**پایپ لاین( Pipeline)**

پایپ لاین : به توالی خطی ای از ماژول های مشخص می گویند که در جهتی مشخص در حال حرکت هستند. خروجی هر ماژول ورودی ماژول بعدی است.

مثلا در دنیای CPU ها، پایپ لاین، به وسیله ای درون CPU گفته می شود که CPU را قادر می سازد دستورات را قبل از اجرا بخواند، و در صورت تایید و به محض تکمیل و بعد از تحویل آن، دستور بعدی را بخواند.

پایپ لاین های مایکروسافت Azure، سرویس های ابری ای(cloud services) هستند که می توانید برای ساخت یا build و test خودکار پروژه های خود استفاده کرده و آنها را برای سایر کاربران در دسترس قرار دهید. این روش برای تمامی زبان های برنامه نویسی و هر نوعی از پروژه قابل اجراست. با Azure Pipelines می توانید از زبانهای زیادی مانند Python ، Java ، JavaScript ، PHP ، Ruby ، ​​C # ، C ++ و Go استفاده کنید.

Azure pipeline دو بخش اصلی دارند :

**پایپ لاین CI/CD یکی از بهترین شیوه های پیاده سازی در تیم های devops به منظور دریافت تغییرات ایجاد شده در کد، بصورت مداوم و مطمئن است.** مفاهیم CI و CD، تجسم یک فرهنگ است. مجموعه ای از قواعد و اقداماتی که تیم های توسعه اپلیکیشن را قادر می سازد تغییرات ایجاد شده در کدها را سریع تر و مطمئن تر بدست مخاطب برساند. پیاده سازی این قواعد و اقدامات را اصطلاحا CI/CD pipeline می گویند. استفاده از CI/CD یکی از بهترین اقداماتی است که تیم های DevOps می توانند انجام دهند.

**1- معنی CI (Continuous Integration) :**

به معنی ادغام مداوم است به عبارت دیگر امروزه که appها در چندین پلتفرم و ابزار های مختلف توسعه پیدا میکنند باید مجموعه ای از اقدامات ایجاد شود که تیم های توسعه را به سمت پیاده سازی تغییرات کوچک و بررسی کدها در ریپازیتوری، آنهم بطور مداوم می برد. لذا نیاز به مکانیزمی برای ادغام (integrate) و تایید(validate ) تغییرات مختلف، اهمیت بالاتری پیدا می کند. پس CI به ادغام مداوم اشاره دارد که فرایندی اتوماتیک برای توسعه دهندگان است. یک CI موفق، باعث خواهد شد تغییرات جدید در کدها، به صورت منظم Build شده، test شده و با یک ریپازیتوری مشترک merge شود. پس CI می تواند بعنوان راه حلی در برابر وجود تعداد زیاد برنچ ها بصورت همزمان باشد، که این امر امکان تداخل برنچ ها با یکدیگر را کاهش می دهد

زمانی که CI در حال اجراست، توسعه دهندگان بطور مداوم کدهایشان را به ریپوزیتوری ورژن کنترل commit می کنند و اکثر تیم ها این استاندارد را دارند که کدها حداقل روزی یک بار باید commit شوند. بنیاد و پایه پشت این کار این است که شناسایی عیوب و سایر مشکلات کیفی نرم افزار در کدهایی که کمتر تغییر یافته اند، آسان تر است. علاوه بر این، زمانی که توسعه دهندگان روی چرخه عمر کمتری از commit ها کار می کنند، احتمال اینکه یک قطعه کد توسط چندین برنامه نویس بصورت همزمان ویرایش شود، کاهش یافته و نیاز به merge در زمان commit کاهش میابد.

فرایند build می تواند بوسیله جمع آوری نرم افزار، دیتابیس و اجزای دیگر بطور اتوماتیک انجام شود. فرض کنید اگر در حال توسعه یک اپ جاوا هستید، CI تمامی فایل های استاتیک وب سرور مانند HTML و CSS و جاوا اسکریپت را همراه با اپ جاوا و تمام اسکریپت های دیتابیس جمع آوری می کند.

CI نه تنها تمام اجزای نرم افزار و دیتابیس را جمع آوری می کند، بلکه این اتوماسیون می تواند unit test ها و سایر تست ها را بر روی آن انجام دهد. این فرایند تست می تواند به توسعه دهندگان آن کد بازخورد دهد که کدهای جدیدی که نوشته اند، نباید در هیچ unit test شکست بخورد.

CI به توسعه دهندگان در ادغام کدهایشان با یک برنچ مشترک یاری می دهد. که این ادغام مرتبا یا گاها روزانه و ساعتی خواهد بود. به محض اینکه تغییرات ایجاد شده توسط توسعه دهنده merge شد، بصورت خودکار با build اپ اعتبار سنجی شده و سطوح مختلف تست های اتوماتیک (معمولا unit و تست های ادغام) روی آن صورت می گیرد تا اطمینان حاصل شود که تغییرات صورت گرفته، باعث بهم ریختگی اپ نمی شود. اگر هرگونه تداخل یافت شود، CI کار رفع عیب را آسان تر و سریع تر می کند.

هدف بنیادی CI، دستیابی به راهی اتوماتیک و استوار برای ساخت (Build) ، پکیج (Package) و تست اپلیکیشن ها است. زمانی که فرایند ادغام باثباتی انجام شود، تیم ها راحت تر و با سرعت بالاتری می توانند تغییراتشان را اعمال کنند و باعث بوجود آمدن همکاری بهتر و کیفیت بالاتر نرم افزار می شود.

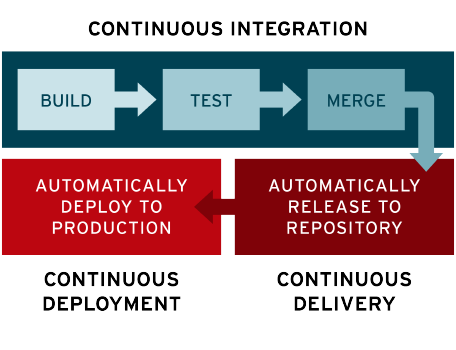
**2- معنی (Continuous Delivery) CD :**

معنی لغوی آن تحویل مداوم میشود. CD زمانی کار CI به اتمام می رسد عملیات رساندن اپلیکیشن ها به محیط های زیرساختی را بطور اتوماتیک انجام می دهد.

بسیاری از تیم ها از چندین محیط توسعه بجای استفاده تنها از محیط production استفاده می کنند. مانند محیط های development و محیط های testing .

CDاطمینان حاصل می کند که راهی اتوماتیک برای رساندن تغییرات کدها به این محیط ها وجود داشته باشد. ابزارهای CI/CD پارامترهای مختص به هر محیط را ذخیره کرده و سپس هر سرویس مورد نیازی که باید در وب سرور ها، دیتابیس ها و ... استفاده شود را در زمان تحویل، صدا می زند.

CD، به تحویل مداوم) (continuous delivery و توزیع مداوم (continuous deployment) اشاره دارد که مفاهیم مرتبط با یکدیگری هستند و برخی اوقات بجای هم استفاده می شوند. اما هر دوی آنها در مورد اتوماتیک سازی stage های پایپ لاین است.

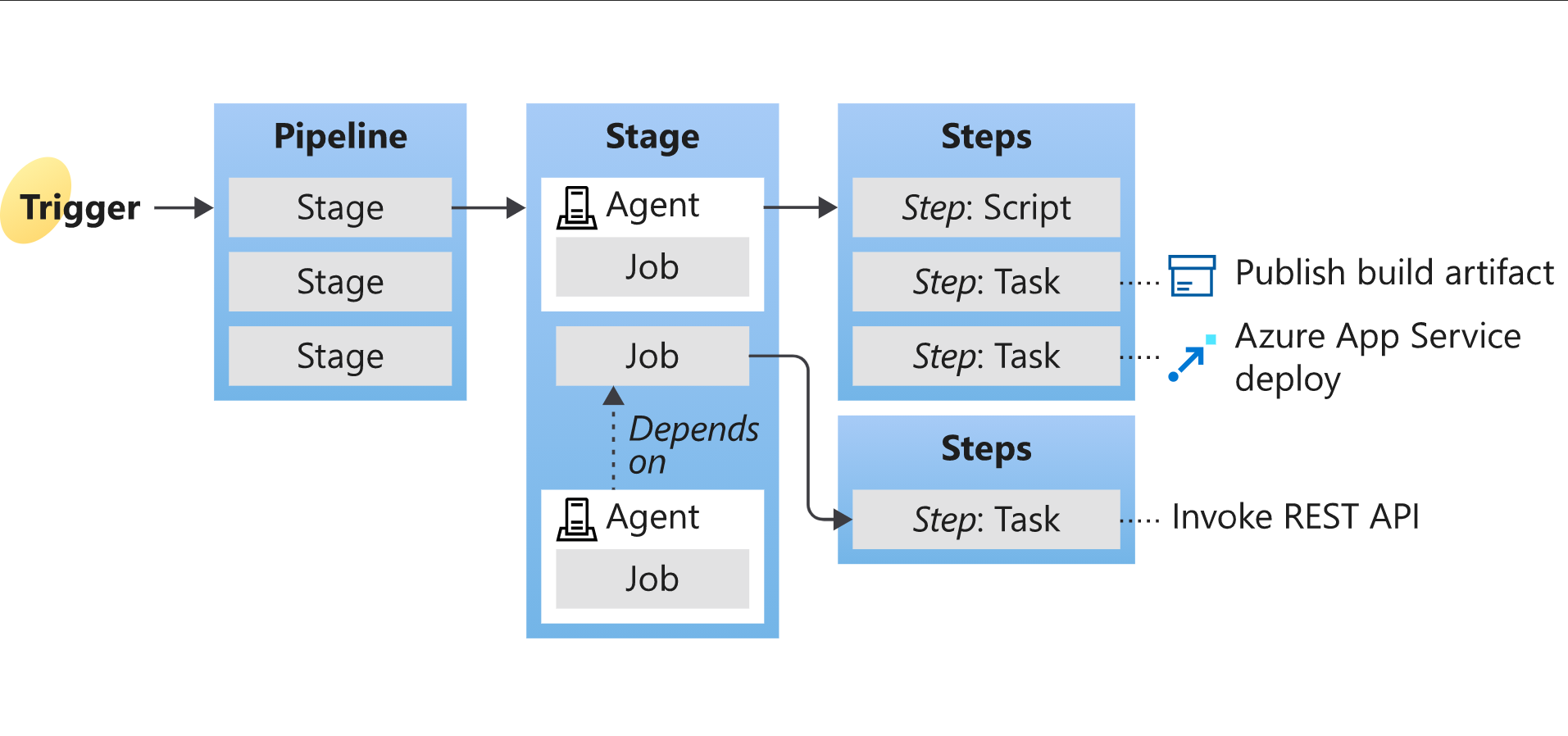
تحویل مداوم بدین معناست که تغییرات کدهایی که توسط برنامه نویس بوجود آمده، به صورت خودکار تست باگ شده و در ریپازیتوری آپلود می شود. جایی که این ریپازیتوری ها می توانند در یک محیط production توزیع شوند. در نهایت، هدف تحویل مداوم، اطمینان از کمترین تلاش ممکن برای استقرار کدهای جدید است.توزیع مداوم، اشاره به انتشار خودکار تغییرات برنامه نویس از ریپازیتوری به production دارد، یعنی به جایی که برای مشتریان قابل استفاده است. پس تیم های اجرایی زمان کمتری نسبت به فرایندهای دستی توزیع محصول به کاربران مصرف می کنند.

Kholaseee:::::::::::

ادغام مداوم (CI)، test و build های خودکار را برای پروژه شما انجام می دهد. CI به شما کمک می کند تا باگ ها را به سرعت و در چرخه توسعه بیابید (نه در زمان توزیع اپ). خروجی تولید شده توسط سیستم های CI را مصنوعات (artifacts ) می گویند که مورد استفاده CD می باشد. CD پس از دریافت از CI، بصورت خودکار کدها را در چندین stage مستقر کرده و تست می کند.

پس بطور خلاصه می توان اینطور گفت که سیستم های CI تولید کننده artifact های قابل deploy هستند که شامل زیرساخت ها و اپ ها می شوند. سپس این artifact ها بصورت خودکار توسط پایپ لاین های انتشار، مصرف شده و نسخه های جدیدی را بر روی مقاصد انتخابی شما ایجاد می کنند.

**2 .** توضیح اجزای تشکیل دهنده ی :: azure pipeline



1. ابتدا Trigger را داریم که تنظیم شده است تا به pipeline بگوید چه زمانی اجرا شود. می توانید pipeline را طوری پیکربندی کنید که با push به repository، در زمان های برنامه ریزی شده یا پس از اتمام build دیگر ، اجرا شود. همه این اقدامات به عنوان trigger شناخته می شوند.
2. Pipeline از یک یا چند stage تشکیل شده. یک Pipeline میتواند در یک محیط(envirement ) یا بیشتر قرار بگیرد (deploy ) .

خط لوله روند CI/CD برنامه شما را تعریف می کند و از یک یا چند مرحله تشکیل شده است. می توان آن را به عنوان یک workflow در نظر گرفت که نحوه اجرای مراحل test, build و deployment شما را مشخص می کند.

1. یک stage راهی برای سازماندهی jobs در Pipeline است و هر مرحله می تواند یک یا چند jobs داشته باشد. یک stage یک مرز منطقی در Pipeline است. می تواند برای نشان دادن تفکیک concern ها (به عنوان مثال Build ، QA و تولید) مورد استفاده قرار گیرد.
2. هر job با یک agent کار می کند. یک job می تواند بدون agent نیز باشد. وقتی build یا deployment شما اجرا شود ، سیستم یک یا چند jobs را شروع می کند. agent در حال محاسبه زیرساخت ها از طریقagent software ای که نصب شده است که همزمان یک job را اجرا می کند.
3. هر jobs با یک agent کار می کند. یک jobs نشان دهنده مرز اجرایی مجموعه ای از steps است. تمام steps با هم بر روی همان agent اجرا می شوند.
4. هر agent یک job را که شامل یک یا چند step است اجرا میکند.step کوچک ترین بخش یک pipeline است که میتواند script یا task باشد
5. task یک بلوک ساختاری برای تعریف اتوماسیون در یک pipeline است. یک packaged script یا procedure است که با مجموعه ای از ورودی ها abstracted شده است.
6. یک اسکریپت با استفاده command line ، PowerShell یا Bash کد را به عنوان یک step در pipeline شما اجرا می کند.. برخلاف task ، یک اسکریپت کد است که مخصوص pipeline شما است.

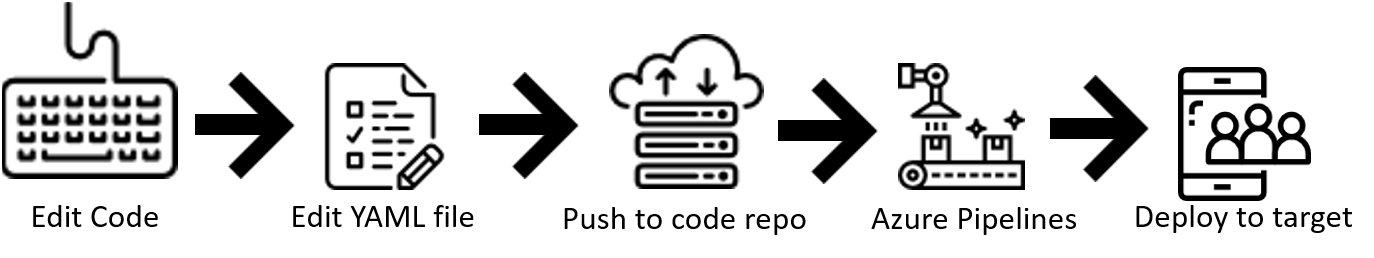
* A [trigger](https://docs.microsoft.com/en-us/azure/devops/pipelines/get-started/key-pipelines-concepts?view=azure-devops#trigger) tells a Pipeline to run.
* A [pipeline](https://docs.microsoft.com/en-us/azure/devops/pipelines/get-started/key-pipelines-concepts?view=azure-devops#pipeline) is made up of one or more [stages](https://docs.microsoft.com/en-us/azure/devops/pipelines/get-started/key-pipelines-concepts?view=azure-devops#stage). A pipeline can deploy to one or more [environments](https://docs.microsoft.com/en-us/azure/devops/pipelines/get-started/key-pipelines-concepts?view=azure-devops#environment).
* A [stage](https://docs.microsoft.com/en-us/azure/devops/pipelines/get-started/key-pipelines-concepts?view=azure-devops#stage) is a way of organizing [jobs](https://docs.microsoft.com/en-us/azure/devops/pipelines/get-started/key-pipelines-concepts?view=azure-devops#job) in a pipeline and each stage can have one or more jobs.
* Each [job](https://docs.microsoft.com/en-us/azure/devops/pipelines/get-started/key-pipelines-concepts?view=azure-devops#job) runs on one [agent](https://docs.microsoft.com/en-us/azure/devops/pipelines/get-started/key-pipelines-concepts?view=azure-devops#agent). A job can also be agentless.
* Each [agent](https://docs.microsoft.com/en-us/azure/devops/pipelines/get-started/key-pipelines-concepts?view=azure-devops#agent) runs a job that contains one or more [steps](https://docs.microsoft.com/en-us/azure/devops/pipelines/get-started/key-pipelines-concepts?view=azure-devops#step).
* A [step](https://docs.microsoft.com/en-us/azure/devops/pipelines/get-started/key-pipelines-concepts?view=azure-devops#step) can be a [task](https://docs.microsoft.com/en-us/azure/devops/pipelines/get-started/key-pipelines-concepts?view=azure-devops#task) or [script](https://docs.microsoft.com/en-us/azure/devops/pipelines/get-started/key-pipelines-concepts?view=azure-devops#script) and is the smallest building block of a pipeline.
* A [task](https://docs.microsoft.com/en-us/azure/devops/pipelines/get-started/key-pipelines-concepts?view=azure-devops#task) is a pre-packaged script that performs an action, such as invoking a REST API or publishing a build artifact.( Invoke REST API check enables you to integrate with any of your existing services. Periodically, make a call to a REST API and continue if it returns a successful response. [Learn More](https://docs.microsoft.com/en-us/azure/devops/pipelines/tasks/utility/http-rest-api?view=azure-devops))
* An [artifact](https://docs.microsoft.com/en-us/azure/devops/pipelines/get-started/key-pipelines-concepts?view=azure-devops#artifact) is a collection of files or packages published by a [run](https://docs.microsoft.com/en-us/azure/devops/pipelines/get-started/key-pipelines-concepts?view=azure-devops#run).

**3. ...بریم به ساختن : azure pipeline**

**Pipeline -> new pipeline**

**1- تعریف پایپ لاین ها توسط سینتکس YAML**

پایپ لاین هایتان را از طریق یک فایل YAML به نام azure-pipelines.yml در کنار سایر فایل های پروژه تعریف می کنید.



* پایپ لاین شما همراه با کدهایتان ورژن داده می شوند. آنها از ساختار مشابهی با branch ها بهره می برند. تغییرات ایجاد شده در کد را مانند قبل، از طریق بازبینی در pull request ها بررسی می کنید.
* می توانید در هر برنچ (شاخه)، قواعد build را با اصلاح فایل azure-pipelines.yml تغییر دهید.
* هر تغییری در فرایند build (مرحله قبل) می تواند مسبب شکست یا ارور شود. اما از آنجائیکه تغییرات در ورژن کنترل رخ می دهند، می توانید به راحتی مشکل را یافته و آن را حل کنید.

این مراحل اساسی را طی کنید:

1- پایپ لاین Azure را پیکربندی کنید تا از ریپازیتوری گیت مورد نظر شما استفاده کند.

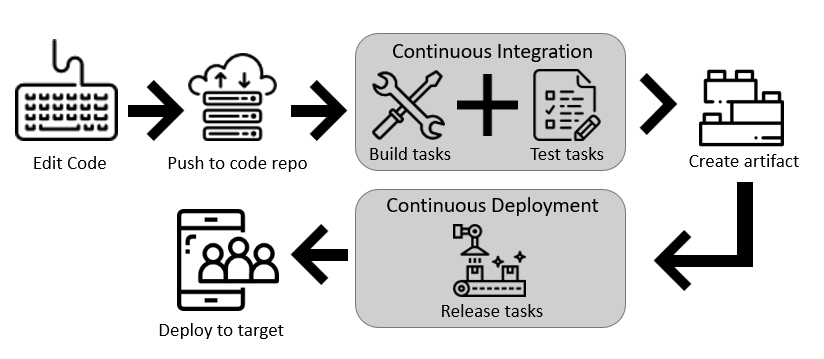
2- فایل azure-pipelines.yml را متناسب با build خود تغییر دهید.

3- کدهایتان را به ریپازیتوری ورژن کنترل خودتان push کنید. این عمل باعث خواهد شد که build و استقرار پروژه شما بر اساس قواعد تعریفی تان بصورت خودکار انجام شده و از این پس می توانید نتایج این اقدام را بررسی (monitor) کنید.

هم اکنون کد شما بروز رسانی شده، build شده، test شده و پکیج شده است. اکنون می توانید آن را به هر مقصدی توزیع کنید!

**2- تعریف پایپ لاین ها از طریق رابط کاربری کلاسیک**

امکان ایجاد و پیکربندی پایپ لاین ها در پورتال وب Azure DevOps با ویرایشگر رابط کاربری کلاسیک نیز وجود دارد. باید یک build pipeline ایجاد کنید تا بتوانید کدهایتان را build و test کنید و سپس artifact ها را منتشر کنید. همچنین یک release pipeline برای مصرف این artifact ها در مقاصد توزیع ایجاد کنید.



مراحل اساسی زیر را طی کنید:

1- پایپ لاین Azure را پیکربندی کنید تا از ریپوزیتوری گیت شما استفاده کند.

2- از ویرایشگر کلاسیک Azure Pipelines بمظور ایجاد و پیکربندی build تان همچنین release pipeline ها استفاده کنید.

3- کدهایتان را به ریپازیتوری ورژن کنترلتان push کنید. با این کار، پایپ لاین ایجاد شده از مراحل قبل را فعال کرده و وظایف را (مانند build و test کد) اجرا می کنید.

بمحض build، باعث ایجاد artifact هایی می شوید که توسط ادامه pipeline برای اجرای وظایف بعدی (مانند توزیع یا staging یا production) استفاده می شوند.

کد شما هم اکنون بروزرسانی شده، build شده، test شده و پکیج شده است. این کد هم اکنون می تواند در هر مقصدی توزیع شود.

برخی از ویژگی های پایپ لاین ها تنها در روش استفاده مستقیم از YAML وجود داشته و برخی دیگر تنها در روش رابط کاربری کلاسیک و یا در هر دو بصورت مشترک وجود دارد. برای مشاهده لیست ویژگی ها و وضعیت پشتیبانی شان توسط هر دو روش به جدول زیر مراجعه کنید:

| **TABLE 2** | | | | |
| --- | --- | --- | --- | --- |
| **Feature** | **YAML** | **Classic Build** | **Classic Release** | **Notes** |
| [Agents](https://docs.microsoft.com/en-us/azure/devops/pipelines/agents/agents?view=azure-devops) | Yes | Yes | Yes | Specifies a required resource on which the pipeline runs. |
| [Approvals](https://docs.microsoft.com/en-us/azure/devops/pipelines/release/approvals/?view=azure-devops) | Yes | No | Yes | Defines a set of validations required prior to completing a deployment stage. |
| [Artifacts](https://docs.microsoft.com/en-us/azure/devops/pipelines/artifacts/artifacts-overview?view=azure-devops) | Yes | Yes | Yes | Supports publishing or consuming different package types. |
| [Caching](https://docs.microsoft.com/en-us/azure/devops/pipelines/release/caching?view=azure-devops) | Yes | Yes | No | Reduces build time by allowing outputs or downloaded dependencies from  one run to be reused in later runs.  In Preview, available with Azure Pipelines only. |
| [Conditions](https://docs.microsoft.com/en-us/azure/devops/pipelines/process/conditions?view=azure-devops) | Yes | Yes | Yes | Specifies conditions to be met prior to running a job. |
| [Container jobs](https://docs.microsoft.com/en-us/azure/devops/pipelines/process/container-phases?view=azure-devops) | Yes | No | No | Specifies jobs to run in a container. |
| [Demands](https://docs.microsoft.com/en-us/azure/devops/pipelines/process/demands?view=azure-devops) | Yes | Yes | Yes | Ensures pipeline requirements are met before running a pipeline stage.  Requires self-hosted agents. |
| [Dependencies](https://docs.microsoft.com/en-us/azure/devops/pipelines/process/stages?view=azure-devops) | Yes | Yes | Yes | Specifies a requirement that must be met in order to run the next job or  stage. |
| [Deployment groups](https://docs.microsoft.com/en-us/azure/devops/pipelines/release/deployment-groups/?view=azure-devops) | Yes | No | Yes | Defines a logical set of deployment target machines. |
| [Deployment group jobs](https://docs.microsoft.com/en-us/azure/devops/pipelines/process/deployment-group-phases?view=azure-devops) | No | No | Yes | Specifies a job to release to a deployment group. |
| [Deployment jobs](https://docs.microsoft.com/en-us/azure/devops/pipelines/process/deployment-jobs?view=azure-devops) | Yes | No | No | Defines the deployment steps. |
| [Environment](https://docs.microsoft.com/en-us/azure/devops/pipelines/process/environments?view=azure-devops) | Yes | No | No | Represents a collection of resources targeted for deployment.  Available with Azure Pipelines only. |
| [Gates](https://docs.microsoft.com/en-us/azure/devops/pipelines/release/approvals/gates?view=azure-devops) | No | No | Yes | Supports automatic collection and evaluation of external health signals prior  to completing a release stage. Available with Classic Release only. |
| [Jobs](https://docs.microsoft.com/en-us/azure/devops/pipelines/get-started/key-pipelines-concepts?view=azure-devops) | Yes | Yes | Yes | Defines the execution sequence of a set of steps. |
| [Service connections](https://docs.microsoft.com/en-us/azure/devops/pipelines/library/service-endpoints?view=azure-devops) | Yes | Yes | Yes | Enables a connection to a remote service that is required to execute tasks in  a job. |
| [Service containers](https://docs.microsoft.com/en-us/azure/devops/pipelines/process/service-containers?view=azure-devops) | Yes | No | No | Enables you to manage the lifecycle of a containerized service. |
| [Stages](https://docs.microsoft.com/en-us/azure/devops/pipelines/get-started/key-pipelines-concepts?view=azure-devops) | Yes | No | Yes | Organizes jobs within a pipeline. |
| [Task groups](https://docs.microsoft.com/en-us/azure/devops/pipelines/library/task-groups?view=azure-devops) | No | Yes | Yes | Encapsulates a sequence of tasks into a single reusable task. If using YAML,  see templates. |
| [Tasks](https://docs.microsoft.com/en-us/azure/devops/pipelines/process/tasks?view=azure-devops) | Yes | Yes | Yes | Defines the building blocks that make up a pipeline. |
| [Templates](https://docs.microsoft.com/en-us/azure/devops/pipelines/process/templates?view=azure-devops) | Yes | No | No | Defines reusable content, logic, and parameters. |
| [Triggers](https://docs.microsoft.com/en-us/azure/devops/pipelines/build/triggers?view=azure-devops) | Yes | Yes | Yes | Defines the event that causes a pipeline to run. |
| [Variables](https://docs.microsoft.com/en-us/azure/devops/pipelines/process/variables?view=azure-devops) | Yes | Yes | Yes | Represents a value to be replaced by data to pass to the pipeline. |
| [Variable groups](https://docs.microsoft.com/en-us/azure/devops/pipelines/library/variable-groups?view=azure-devops) | Yes | Yes | Yes | Use to store values that you want to control and make available across multiple  pipelines. |

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

(YAML) (هنوز زبان نشانه‌گذاری نیست) یک زبان برای طبقه‌بندی داده قابل خواندن برای انسان است. معمولاً برای  [configuration files](https://en.wikipedia.org/wiki/Configuration_file) (فایل‌های پیکربندی ) استفاده می‌شود، اما می‌تواند در کاربردهای دیگری هم مورد استفاده قرار گیرد (به عنوان مثال خروجی اشکال‌زدایی) یا انتقال (به عنوان مثال هدر سند). بسیار شبیه به XMLو JSONاست .

به دو روش نوشته می‌شود اولی سبک زبان پایتون و دومی روش فشرده‌تری است که مانند JSON برای لیست‌ها از [] و برای مپ‌ها از {} استفاده می‌کند که YAML 1.2 را زیر مجموعه ی JSONقرار می‌دهد.

کد زیر یک فایل yaml است :

---

doe: "a deer, a female deer"

ray: "a drop of golden sun"

pi: 3.14159

xmas: true

french-hens: 3

calling-birds:

  - huey

  - dewey

  - louie

  - fred

xmas-fifth-day:

  calling-birds: four

  french-hens: 3

  golden-rings: 5

  partridges:

    count: 1

    location: "a pear tree"

  turtle-doves: two

همانطور که میبینید فایل با سه تا dash شروع شده که این خط تیره ها نشان دهنده آغاز document جدید YAML است. YAML از چندین document پشتیبانی می کند . بعد بیشتر این کدها را جفت های key-value تشکیل میدهند مثلا Doe کلیدی است که به مقدار رشته (string) اشاره می کند . فایل با شش جفت key-value شروع می شود که چهار نوع داده مختلف دارند. Doe و ray رشته (string) هستند. Pi یک عدد floating-point است. Xmas یک Boolean است. French-hens یک عدد صحیح(integer) است.

می توانید رشته ها (string) را به صورت نقل قول های تک یا مضاعف یا اصلا بدون نقل قول محصور کنید.( single or double-quotes or no quotes) . YAML اعداد بدون نقل را به عنوان اعداد صحیح یا نقطه شناور(integers or floating point.) تشخیص می دهد.

مورد هفتم یک آرایه است. Calling-birds دارای چهار عنصر است که هر کدام با یک خط تیره (opening dash) مشخص می شوند.

سرانجام ، ما xmas-fifth-day را می بینیم که پنج عنصر دیگر در داخل خود دارد ،می توانیم xmas- fifth -day را به عنوان دیکشنری مشاهده کنیم که شامل دو رشته string ، دو عدد صحیح integer و دیکشنری دیگر است.

YAML supports nesting of key-values, and mixing types.

توجه : key:(space)value فاصله مهم

معادل کد بالا در JSON :

{

 "doe": "a deer, a female deer",

 "ray": "a drop of golden sun",

 "pi": 3.14159,

 "xmas": true,

 "french-hens": 3,

 "calling-birds": [

    "huey",

    "dewey",

    "louie",

    "fred"

 ],

 "xmas-fifth-day": {

 "calling-birds": "four",

 "french-hens": 3,

 "golden-rings": 5,

 "partridges": {

   "count": 1,

   "location": "a pear tree"

 },

 "turtle-doves": "two"

 }

}

JSON و YAML از قابلیت های مشابهی برخوردار هستند و شما می توانید اکثر document ها را بین format ها تبدیل کنید.

Whitespace در YAML : Whitespace بخشی از قالب بندی YAML است. خطوط جدید پایان یک قسمت را نشان می دهد .

شما یک سند YAML را با فرورفتگی ساختار می دهید. سطح تورفتگی می تواند یک یا چند فاصله باشد.tab ها ممنوع اند زیرا ابزار با آنها متفاوت رفتار می کند.

این document را در نظر بگیرید. item های داخل stuff با دو فاصله ( space ) فرورفتگی دارند.

foo: bar

    pleh: help

    stuff:

      foo: bar

      bar: foo

بیایید نگاهی به نحوه نشان دادن یک اسکریپت ساده پایتون به این document بیندازیم. ما آن را به عنوان file ای به نام foo.yaml ذخیره خواهیم کرد.

بسته PyYAML جریان فایل YAML را در یک dictionary ،map می کند. ما از بیرونی ترین مجموعه keys and values ، iterate خواهیم کرد و key و string هر valueرا چاپ خواهیم کرد. در اینجا می توانید processor برای پلتفرم مورد علاقه خود پیدا کنید.

import yaml

if \_\_name\_\_ == '\_\_main\_\_':

   stream = open("foo.yaml", 'r')

   dictionary = yaml.load(stream)

   for key, value in dictionary.items():

       print (key + " : " + str(value))

خروجی به صورت :

foo : bar

pleh : help

stuff : {'foo': 'bar', 'bar': 'foo'}

وقتی به پایتون می گوییم یک dictionary را به عنوان رشته چاپ کند ، از inline syntaxا ستفاده می کند که در زیر خواهیم دید. از خروجی می فهمیم که document ما یک دیکشنری پایتون است که دارای دو رشته است و dictionary دیگری درون آن قرار دارد.

simple nesting ساده YAML به ما قدرت ساخت object های پیچیده را می دهد.

Comments :

Comment ها با علامت پوند ( # ) شروع می شود. آنها می توانند پس از یک document value ظاهر شوند یا یک خط کامل را اشغال کنند.

\_\_\_

# This is a full line comment

foo: bar # this is a comment, too

Comments برای انسان است. پردازنده های YAML آنها را کنار می گذارند.

مقادیر موجود در جفتهای key-value در YAML ، scalar هستند. آنها در زبانهایی مانند Perl ، Javascript و Python مانند انواع اسکالر عمل می کنند. معمولاً بهتر است که رشته ها را در " " بگذارید ، اعداد را بدون " " بگذارید تا parser آن را بفهمد.

Key-Value Pairs and Dictionaries

key-value عنصر اصلی ساخت YAML است. هر item در یک سند YAML عضو حداقل یک dictionary است. key همیشه یک string است. value یک scalar است بنابراین می تواند هر نوع داده ای باشد. بنابراین ، همانطور که قبلاً دیدیم ، value می تواند یک رشته ، یک عدد یا یک dictionary دیگر باشد.

انواع عددی (Numeric types ) :

YAML انواع عددی را تشخیص می دهد. ما در بالا floating point و integers را دیدیم. YAML چندین نوع عددی دیگر را پشتیبانی می کند.

یک عدد صحیح می تواند decimal, hexidecimal یا octal باشد.

---

foo: 12345

bar: 0x12d4

plop: 023332

همانطور که انتظار دارید ، Ox نشان می دهد که یک مقدار hex است ، و 0 هم یک مقدار octal را نشان می دهد.

که در پایتون به صورت زیر است :

foo : 12345

bar : 4820

plop : 9946

YAML از اعداد اعشاری floating point و fixed point پشتیبانی می کند.

---

foo: 1230.15

bar:  12.3015e+05

که مقادیر زیر را دارند :

foo : 1230.15

bar : 1230150.0

ما می توانیم عدد یک NAN یا بی نهایت را نیز نشان دهیم :

---

foo: .inf

bar: -.Inf

plop: .NAN

foo بی نهایت است. bar بی نهایت منفی است ، و plop، NAN است.

Strings :

رشته های YAML یونیکد هستند. در بیشتر مواقع ، نیازی نیست که آنها را در " " ها مشخص کنید.

---

foo: this is a normal string

برنامه آزمون ما این را به صورت زیر پردازش می کند:

foo: this is a normal string

اما برای اطمینان ، باید از " " استفاده کنیم:

---

foo: "this is not a normal string\n"

bar: this is not a normal string\n

YAML فقط اولین مقدار را پردازش می کند. از آنجا که مقدار دوم نقل نشده است ، YAML \ n را به عنوان دو کاراکتر در نظر می گیرد.

foo: this is not a normal string

bar : this is not a normal string\n

YAML رشته ها با یک " را هم میفهمد ، اما یک " از تفسیر محتوای رشته به عنوان document formatting جلوگیری می کنند.

مقادیر رشته می تواند بیش از یک خط باشد. با کاراکتر fold (<) می توانید یک رشته را در یک بلوک مشخص کنید.

bar: >

 this is not a normal string it

 spans more than

 one line

 see?

اما بدون خطوط جدید تفسیر می شود.

bar : this is not a normal string it spans more than one line see?

کاراکتر بلوک (لوله) عملکرد مشابهی دارد ، اما YAML فیلد را دقیقاً همانگونه تفسیر می کند.

bar: |

 this is not a normal string it

 spans more than

 one line

 see?

بنابراین ، ما خطوط جدید را در جایی که در document هستند مشاهده می کنیم.

bar : this is not a normal string it

spans more than

one line

see?

Nulls :

شما با یک tilde (~) یا رشته null بدون نقل قول نولها را وارد می کنید.

---

foo: ~

bar: null

برنامه چاپ می کند:

foo : None

bar : None

Python ، null را none چاپ میکند .

Booleans :

YAML مقادیر boolean را با کلمات کلیدی True ، On و Yes برای true نشان می دهد. False با False ، Off یا No. نشان داده می شود.

---

foo: True

bar: False

light: On

TV: Off

Arrays : ( list) (برای ذخیره کردن تعدادی object از یک نوع استفاده میشوند)

می توانید آرایه ها یا لیست ها را روی یک خط مشخص کنید.

---

items: [ 1, 2, 3, 4, 5 ]

names: [ "one", "two", "three", "four" ]

یا می توانید آنها را روی چندین خط قرار دهید.

---

items:

 - 1

 - 2

 - 3

 - 4

 - 5

names:

 - "one"

 - "two"

 - "three"

 - "four"

قالب چند خطی برای لیست هایی مفید است که به جای اسکالر ها شامل objects های پیچیده هستند.

\_\_\_

items:

 - things:

     thing1: huey

     things2: dewey

     thing3: louie

 - other things:

     key: value

یک آرایه می تواند حاوی هر مقدار معتبر YAML باشد. مقادیر موجود در یک لیست لازم نیست که یک نوع باشند.

Dictionaries : (برای ذخیره کردن انواع داده های مربوط به یک object، داده ها میتوانند انواع مختلفی داشته باشند)

مانند آرایه ها ، می توانید Dictionarie ها را به صورت درون خطی ( inline ) قرار دهید. ما این format را در بالا دیدیم. نحوه چاپ Dictionarie توسط پایتون :

---

foo: { thing1: huey, thing2: louie, thing3: dewey }

که میتوانند مانند :

---

foo: bar

bar: foo

و البته ، آنها می توانند تو در تو( nested) قرار بگیرند و هر ارزشی را نگه دارند :

---

foo:

 bar:

   - bar

   - rab

   - plop

Chomp Modifiers :

مقادیر چند خطی (Multiline values ) ممکن است با فضای خالی خاتمه یابد و بسته به نحوه پردازش document ممکن است نخواهید آن را حفظ کنید. YAML دارای نوار chomp و عملگرهای chomp است.

برای ذخیره آخرین کاراکتر ، یک plus (+) به دستورات fold (<) یا block (|) اضافه کنید.

bar: >+

 this is not a normal string it

 spans more than

 one line

 see?

بنابراین ، اگر این مقدار با فضای خالی به پایان برسد ، مانند یک خط جدید ، YAML آن را حفظ می کند.

bar: |-

 this is not a normal string it

 spans more than

 one line

 see?

YAML یک زبان قدرتمند است که می تواند برای configuration files ، پیام های بین برنامه ها و صرفه جویی در state برنامه استفاده شود.

**4.variable :**

Variablesراهی راحت برای قرار دادن بیت های کلیدی داده در قسمتهای مختلف خط لوله به شما می دهند. متداولترین کاربرد Variablesتعریف یک مقدار و استفاده ار آن در pipeline است. همه variables به صورت strings ذخیره می شوند و قابل تغییر هستند. مقدار یک متغیر می تواند از اجرا به اجرا یا job به job دیگر pipeline شما تغییر کند.

وقتی Variablesیکسانی را در چندین مکان با همان نام تعریف کنید ، بیشترین Variablesدر سطح locall برنده است. بنابراین ، یک متغیر تعریف شده در سطح job می تواند یک Variablesتنظیم شده در سطح stage را override کند. یک Variablesتعریف شده در سطح stage ، Variablesرا که در سطح pipeline root قرار دارد ، override میکند. یک Variablesتنظیم شده در سطح pipeline root ، یک متغیر را در UI تنظیمات خط لوله override می کند.

Variablesبا runtime parameters متفاوت هستند که در هنگام template parsing. تایپ می شوند و در دسترس قرار میگیرند.

1. User-defined variables:

هنگامی که یک متغیر را تعریف می کنید ، می توانید از syntaxes مختلف (macro, template expression, or runtime ) استفاده کنید و اینکه از چه syntaxes استفاده می کنید تعیین می کند که متغیر شما در کجای pipeline قرار گیرد.

در خطوط لوله YAML ، می توانید متغیرها را در سطح root, stage, job تنظیم کنید. همچنین می توانید متغیرهایی را خارج از خط لوله YAML در UI تعیین کنید. وقتی یک متغیر را در UI تنظیم می کنید ، آن متغیر می تواند رمزگذاری شود و به صورت secret تنظیم شود. Secret variables به طور خودکار در خطوط لوله YAML رمزگشایی نمی شوند و باید به YAML file شما با env: یا یک متغیر در سطح root منتقل شوند.

User-defined variables را می توان فقط برای خواندن تنظیم کرد.( read-only)

برای در دسترس قرار دادن متغیرها در چندین pipelines می توانید از یک variable group استفاده کنید.

برای تعریف متغیرهایی که در چندین pipelines در یک file استفاده می شوند می توانید از templates استفاده کنید.

2. System variables :

علاوه بر User-defined variables ، Azure Pipelines دارای . System variables با مقادیر از پیش تعریف شده است. YAML وclassic build pipelines هر دو از predefined variables استفاده می کنند و classic release pipelines از release variable استفاده می کنند. که توضیحات predefined :

<https://docs.microsoft.com/en-us/azure/devops/pipelines/build/variables?view=azure-devops&tabs=yaml>

توضیح release:

<https://docs.microsoft.com/en-us/azure/devops/pipelines/release/variables?view=azure-devops&tabs=batch>

هنگام اجرای خط لوله ، System variables با مقدار فعلی خود تنظیم می شوند. برخی از متغیرها به طور خودکار تنظیم می شوند. شما به عنوان نویسنده خط لوله یا کاربر نهایی ، مقدار یک system variable را قبل از اجرای خط لوله تغییر می دهید.

. System variables ، read-only هستند.

3. Environment variables :

Environment variable ها خاص OS مورد استفاده شما هستند. آنها از طریق روشهای platform-specific به یک خط لوله تزریق می شوند. format مربوط به Environment variable مربوط به سیستم عامل خاص برنامه نویسی(scripting platform ) شما است.

در سیستم های UNIX macOS and Linux)) Environment variable دارای قالب NAME$ هستند. در ویندوز ، قالب٪ NAME٪ برای batch و $env:NAME در PowerShell است

System and user-defined variables همچنین به عنوان Environment variable به platform شما تزریق می شوند. وقتی متغیرها به Environment variable تبدیل می شوند ، نام متغیرها بزرگ می شوند و . به \_ تبدیل می شوند. به عنوان مثال ، نام متغیر any.variable نام متغیر ANY\_VARIABLE$ می شود.

# Set variables once

variables:

configuration: debug

platform: x64

steps:

# Use them once

- task: MSBuild@1

inputs:

solution: solution1.sln

configuration: $(configuration) # Use the variable

platform: $(platform)

# Use them again

- task: MSBuild@1

inputs:

solution: solution2.sln

configuration: $(configuration) # Use the variable

platform: $(platform)

**سطح دسترسی variables :**

variables:

global\_variable: value # this is available to all jobs

jobs:

- job: job1

pool:

vmImage: 'ubuntu-16.04'

variables:

job\_variable1: value1 # this is only available in job1

steps:

- bash: echo $(global\_variable)

- bash: echo $(job\_variable1)

- bash: echo $JOB\_VARIABLE1 # variables are available in the script environment too

- job: job2

pool:

vmImage: 'ubuntu-16.04'

variables:

job\_variable2: value2 # this is only available in job2

steps:

- bash: echo $(global\_variable)

- bash: echo $(job\_variable2)

## - bash: echo $GLOBAL\_VARIABLE

## Set secret variables

<https://docs.microsoft.com/en-us/azure/devops/pipelines/process/variables?view=azure-devops&tabs=yaml%2Cbatch>

## Environment

An environment is a collection of resources, where you deploy your application. It can contain one or more virtual machines, containers, web apps, or any service that's used to host the application being developed. A pipeline might deploy the app to one or more environments after build is completed and tests are run.

## Approvals

[Approvals](https://docs.microsoft.com/en-us/azure/devops/pipelines/process/approvals?view=azure-devops) define a set of validations required before a deployment can be performed. Manual approval is a common check performed to control deployments to production environments. When checks are configured on an environment, pipelines will stop before starting a stage that deploys to the environment until all the checks are completed successfully.

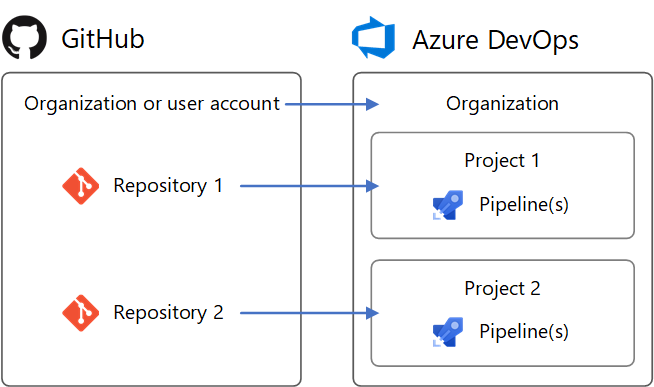
## Deployment group

A deployment group is a set of deployment target machines that have agents installed. A deployment group is just another grouping of agents, like an agent pool. You can set the deployment targets in a pipeline for a job using a deployment group. Learn more about provisioning agents for [deployment groups](https://docs.microsoft.com/en-us/azure/devops/pipelines/release/deployment-groups/howto-provision-deployment-group-agents?view=azure-devops).

ریپازیتوری هایی که درpipeline ساپورت میشود:

| **Repository type** | **Azure Pipelines (YAML)** | **Azure Pipelines (classic editor)** | **Azure DevOps Server 2019, TFS 2018, TFS 2017, TFS 2015.4** | **TFS 2015 RTM** |
| --- | --- | --- | --- | --- |
| [Azure Repos Git](https://docs.microsoft.com/en-us/azure/devops/pipelines/repos/azure-repos-git?view=azure-devops) | Yes | Yes | Yes | Yes |
| [Azure Repos TFVC](https://docs.microsoft.com/en-us/azure/devops/pipelines/repos/tfvc?view=azure-devops) | No | Yes | Yes | Yes |
| [GitHub](https://docs.microsoft.com/en-us/azure/devops/pipelines/repos/github?view=azure-devops) | Yes | Yes | No | No |
| [GitHub Enterprise Server](https://docs.microsoft.com/en-us/azure/devops/pipelines/repos/github-enterprise?view=azure-devops) | Yes | Yes | TFS 2018.2 and higher | No |
| [Bitbucket Cloud](https://docs.microsoft.com/en-us/azure/devops/pipelines/repos/bitbucket?view=azure-devops) | Yes | Yes | No | No |
| [Bitbucket Server](https://docs.microsoft.com/en-us/azure/devops/pipelines/repos/on-premises-bitbucket?view=azure-devops) | No | Yes | Yes | Yes |
| Subversion | No | Yes | Yes | No |

* An Azure DevOps **organization** for your GitHub **organization or user account**
* Azure DevOps **Projects** for your GitHub **repositories**



**سیستم های ورژن کنترل**

قبل از اینکه بخواهید از CI/CD برای اپ های خود استفاده کنید، باید کد های سورس خود را در یک سیستم ورژن کنترل قرار دهید. پایپ لاین های Azure با سرویس های Github، Azure Repos Git & TFVC، Bitbucket Cloud و ... سازگاری دارد. **V**ersion **C**ontrol **S**ystem که به اختصار **VCS** خوانده می‌شود عبارت است از سیستمی که به توسعه‌دهندگان نرم‌افزار کمک می‌کند تا علاوه بر امکان مشارکت روی پروژه‌های نرم‌افزاری، بتوانند به تاریخچه‌ای از کدهایی که قبلاً نوشته‌اند نیز دست پیدا کنند و به طور کلی اهداف استفاده از سیستم‌های ورژن کنترل (VCS) را می‌توان در موارد زیر خلاصه نمود:

- فراهم آوردن فرصتی برای توسعه‌دهندگان به منظور کار کردن به صورت هم‌زمان  
- مجزاسازی نسخه‌های توسعه داده شدهٔ اختصاصی تک‌تک توسعه‌دهندگان  
- نگهداری تاریخچه‌ای از هر نسخه از هر چیزی که به اشتراک گذاشته شود

به طور کلی، می‌توانیم سیستم ورژن کنترل را به عنوان یک دیتابیس در نظر بگیریم که به توسعه‌دهندگان این اجازه را خواهد داد تا در هر زمانی که تمایل داشته باشند، نسخه‌ای از نرم‌افزار مد نظر خود را در آن ذخیره سازند. حال در آینده زمانی که به یکسری Version (نسخه) قدیمی نگاهی می‌اندازیم، دقیقاً متوجه خواهیم شد که کدام بخش از نرم‌افزار دستخوش تغییر شده است. به طوری کلی، از جمله مزایای نرم‌افزارهای ورژن کنترلی مثل Git این است که محدود به زبان برنامه‌نویسی خاص و همچنین ویرایشگر کد خاصی نبوده و هر نوع سورس‌کدی که با هر نرم‌افزاری نوشته شده باشد را ساپورت می‌کنند

**مقصد های توزیع(deploy)**

از Azure Pipelines برای deploy کدهایتان در چندین مقصد می توانید استفاده کنید. این مقاصد می تواند شامل container registries، virtual machines، Azure Services یا هر مقصد تحت کلود دیگری باشد.

Package formats

برای تولید بسته هایی که توسط دیگران قابل مصرف است ، می توانید بسته های NuGet ، npm یا Maven را در مخزن مدیریت بسته داخلی (built-in package management repository)در Azure Pipelines منتشر کنید. همچنین می توانید از مخزن مدیریت بسته دیگر به انتخاب خود استفاده کنید.