Table of Contents

Introduction

Evergreen Delivery IP is ACAI CTO driven program to improve the way CTO and field teams are finding, reusing, becoming contributors and leading evolution of managed and community IP.

Goals

Specific high level goals of the program are:

Harmonize processes, Taxonomy and tools used

- Simplify through unification of process and tools
- Use modern delivery principles in IP management
- Allow for better composition into larger IP elements across teams

Reduce IP Fragmentation

- Drive innovations back to original IP
- Achieve faster IP evolution
- Build larger assets using existing blocks, do not “fork”

Drive More Community Involvement and Leadership

- Grow grassroots contribution for IP
- Use SAO team as supporting force, not always as leading one
- Leverage strength of entire MCS to build and improve IP

More About Initiative

More on the reasons for the evergreen IP work and the needs that it tries to address, see in:

- Why IP?
- Why change anything? (aka challenges and needs identified)

More on the planned changes and horizon plan see in:

- Directions of Change
- Objectives
- Horizons (including the North star)

Why IP?

There is no real question of why IP when it comes to Services engagements. Reusing IP in Services engagements allow for:

- better positioning
- less effort/cost
- higher quality
- expected value

Better Positioning

Existing experience allows better sales positioning with customers. Running sales campaigns aligned with IP enables scale.

Less Effort/Cost

One of the main elements is use of less effort in pre-sales and delivery, reducing time to proposal or having shorter delivery timeframes.

Higher Quality

Reuse of previous experience avoids the same problems, following proven practices results in higher quality and customer satisfaction.

Expected Value

Customers engaging MCS still expect MS global experience and existing “products” to be used instead of pure custom system integration work. Reuse enables meeting this expectation and charging our premium.

Why change anything?

Although IP management has been something that has been done for as long as we have Services organization in Microsoft, when we look at evolution of approach and our teams, different IP assets we are creating and projects we are delivering, there are challenges and needs identified that need to be addressed. These are grouped in 4 categories below.

Governance of IP Lifecycle

- no unified approach and clear responsibilities across all ACAI teams,
- different success levels of involving community,
- limited contribution back after reuse – feedback and updates (maintenance costs up, no single truth)
- weak insights into reuse, contribution, customer value
- IP fragmentation

Discovery/reuse by field

- hard to consume content,
- need better support for sellers and pre-sales architects,
- high time to first estimate,
- wrong assumptions around IP depth + readiness resulting in delivery issues

Consistency

- lack of consistency within previous domains/teams,
- use of different tools hindering benefits in larger projects
- different feedback mechanisms
- no consistent requirements capturing
- no consistent delivery methodology

Composition

- large projects need to realign/reassemble IP if many packages are brought in
- lack of consistency of guiding principles & constraints in BOM/architecture to drive easier composition at time of delivery

Directions of Change

Given that improvement of delivery IP process is a huge endeavor, the evergreen delivery IP initiative focuses on four specific **directions of change** as described below. Each of these directions of change then are expanded via specific objectives.

Consistency Across Domain

Leverage uniform approaches, clear set of tools across entire domain to drive simplicity and consistency in how IP is “developed”, managed and reused, introduce clear continuum from community to managed IP

Insights Based Evolution

of IP

Get insights on how customer value, contribution and reuse is measured to drive investment and evolution of the IP asset.

of IP Process

Measure and adjust the process of evergreen IP delivery, learn on what works and what does not, where more focus and investment is needed to drive overall goals and specific objectives of the initiative.

Community Involvement

Have community front and center in delivery IP capturing/development/evolution, evolve culture to remove barriers for productive and sustained community involvement throughout IP lifecycle

More Discovery and Reuse

Make sure any ACAI employee can easily discover delivery IP, leverage it in pre-sales and delivery engagements, provide feedback and start contributing regardless of IP type, team behind it and its depth level

Objectives

Objectives are linked to he directions of change. This is why below list of objectives is grouped based on the four directions of change.

Consistency Across Domains

Objectives to drive direction of consistency:

1. Uniform IP content approach
2. One consistent set of IP processes and tools
3. Modern delivery principles leveraged

Insights Based Evolution

Objectives for insights based evolution:

1. IP impact tracking: internal and external insights
2. Delivery IP program evolution based on insights

Community Involvement

Objectives for community involvement:

1. Community in leading role for delivery IP
2. Community involvement sustained
3. Partner and customer involvement enabled

More Discovery and Reuse

Objectives for driving more discovery and reuse:

1. Reuse and contribution culture focus
2. Reduced IP fragmentation
3. Well maintained docs & readiness, reuse and contribution guidance

Horizons

Overall vision that the evergreen delivery IP is contributing towards is to have IP and co-innovation (inside Services, with PG, with customers and partners) at the core of our approach.

Figure below shows the three horizons plan with first level 1 goal being consistency in capturing and discovering IP, second being more grassroots and community involvement and third one being transformation to more IP and co-innovation based approach.

- **The Vision:** co-innovating with partners and customers, leveraging all learnings out of our engagements in a consistent way, enabling grassroots IP contribution and growth
- **Horizon 3:** significant cultural change to enable inner source based internal and external (partners/customers) co-innovation of Services IP, most projects either leverage existing IP or create new one or both, dedicated resource pods working in top investment areas to sustain IP and improve it over time
- **Horizon 2:** processes and tools, investment model created to enable pockets of excellence in field creating IP together with customers and partners, sustaining it over time and driving project reuse through metrics and customer feedback
- **Horizon 1:** enabling uniform approach for managing various types of delivery IP across domain to drive ease of use in finding, reusing and contributing back; metrics-based assessment, SAO teams example of adoption

Overall Approach

Added for testing

This section defines the overall approach suggested for the IP development, innersource community building and discovery of delivery IP. It covers these three main areas looked at at part of the Evergreen Delivery IP initiative:

- team model that is based on the innersource approach, suggests the team model starting from defining the core team, describing the approach for managing contribution and discovery, it also touches on the tools suggested for use to implement the approach
- modular ip asset management that should support diverse teams working on different set of IP assets (documentation, reference architecture, data models, scripts, code and components) and is based on the concept of "content lenses"
- sustaining and growing ip contribution culture that includes approaches suggested as part of the program to ensure contribution moves from current spots of IP contribution successes to more pervasive focus on IP contribution all up in ACAI.

Principles

The main principles followed by the approach are:

- **gradual adoption** to ensure that teams can light up their existing IP assets (repos teams) with Evergreen Delivery approaches and that teams can start their adoption small and then grow as needed
- **flexibility of adoption** to support different approaches by different teams based on their needs, for example:
  - being able to roll out own documentation portal following guidance of the Evergreen Delivery IP starter kit guidance (more control, yet reuse of common approach and practices) or committing to centralized tools that get the documentation sources and host the documentation automatically via centrally managed tool set (less control, but lower barrier of entry)
  - choosing the lenses that are important for the IP in question and committing only to their specific BOM approach allowing other assets to be managed as the team wants
- **distributed sites/repos** are supported by the approach given that there are different tools and repos used today and there is no way this is going to change in the future; as a an example the tools that index the Evergreen IP repos are expected to enable registration of multiple distributed locations where repos are and ensure these all repos can be indexed
- **priority for repo based content** over use of SharePoint especially for the content that is not binary (Word documents etc) to ensure that full tracking of the collaboration is possible and all changes are visible, thus promoting the innersource openness
- **insights and feedback based improvement** based on innersource principles for the Evergreen Delivery IP initiative itself so that teams adopting the principles and working on their specific needs can contribute back common elements that would become more useful for others as well.

Approach Overview Deck

Teams Following Evergreen Delivery

To ensure ACAI teams have as easy as possible way of adoption of the principles and approaches defined in this section overall evergreen delivery IP approach relies on implementation guidance that is enhanced based on core team priorities, but

even more so - based on those implementing approach and providing back feedback and contributions for new tools, lenses and best practices.

Team Model

Team model for the evergreen delivery IP based IP or incubation project is based on the principles of innersource core teams. This page provides more details about the model and main "tools" that the team is leveraging when working on the IP.

Model

The model schematically is shown in the image below.

Image shows three main groups of contributors/users of IP:

- **core team** that consists of the key team members accountable for the IP development and maintenance over time, welcoming external contributors, defining goals and planning the IP evolution based on the needs of customers, issues submitted. In broader innersource community this group of team members are also called "Trusted Contributors".
- **contributors** that are other Microsoft (or with time even partner and customer representatives involved in co-innovation) that are contributing to various aspects of the IP - documentation, components, issues etc. Contributors can over time become core team members.
- **interested parties** that are reusing the IP, providing feedback and over time can become contributors and core team members

Lifecycle

The typical progression/lifecycle of the team working on an IP asset includes these steps:

1. Starts with the core team (can be field team or SAO team) that wants to capture assets
2. Team creates an IP repository and associated IP forum with predefined starter structure and metadata
3. Metadata drives discovery of the IP in existing and upcoming platforms – SEE, Chrysalis
4. Interested parties start to discover IP and as it becomes more popular, contributions grow, contributor and core team grows

Tools

The IP team relies on three main sets of tools to manage the IP and process around IP evolution:

- **Core repository** that contains structured IP content in a version controlled environment (GitHub/ADO). This environment is also used for public discussions, tracking issues, plans (backlog)
- **Discussions** that are used to have meetings and informal information exchange that can be private within core team or public to address entire set of contributors and interested parties. This environment is leveraging MS Teams as the base platform and private channels concept to ensure certain discussions remain scoped at core team only. SharePoint site associated with Teams can also be used to manage binary IP assets that are not supported well in Git based repositories.
- **IP Discovery Catalogue(s)** that are used as the means of discovery of IP. They can be reactive (like Campus) where users need to search for repos/IP they are interested or proactive (as SEE) that provides IP based suggestions based on opportunities/projects users are engaged in. The approach ensures that over time more catalogues and publishing tools can be added without affecting the core team's work.

Diverse Delivery IP

ACAI domain teams work on different IP assets that can range from documentation of processes to scripts and code components that are used in deploying customer solutions. Due to this variety of different assets, one size (approach) cannot fit all teams.

Also it is important that existing assets that have built some time ago can gradually be on-boarded and start benefitting from evergreen delivery IP elements over time.

To ensure this flexibility and modularity of the approach, also allow to start small and grow as IP grows, the evergreen delivery IP introduces concept of content lenses. More on principles, examples of lenses and how it helps overall approach see more in subsections below.

Principles & Process

- The first thing that any IP repository needs is to have **readme** and **contribution** guideline files, and the main **evergreen IP metadata file** describing the project/IP, its team, contribution rules. This already should help potential contributors to find the repository, understand the goal of the project and start reusing, over time contributing back.
- As repository grows and specific additional types of content get deliver, core team behind the IP can decide to commit to certain type of content to be added, for example, to add easy to consume documentation about the IP, APIs it exposes (if this is a component being implemented) etc.
    - This is achieved through "content **lenses**" to ensure that there is a unified approach on how different types of IP assets get managed and described in ACAI.
    - Each lense defines its specific approach for content and metadata describing this content, each lens as part of Evergreen Delivery IP can come with guidance on how to implement it, tools providing additional value. For example, documentation lens defines includes guidance on content authoring, publishing as part of MS Internal site, has Azure pipeline assets to support implementation.

Illustration of starting with base repo and then adding in more lenses depending on type of assets managed in the repo is provided below.

Examples of Lenses

Lenses require additional predefined BOM per type, can enable tools and additional guidance (these BOM items are owned by respective team, Evergreen Delivery IP program only helps with tooling and approach):

- Documentation - documentation lens is helpful for projects that document best practices, patterns, approaches, methodology etc.
- Reference architecture - reference architecture lens includes specific assets on top of general documentation lens to include specific BOM items that are helpful when defining reference architecture, also the metadata associated with reference architecture can help ling it to various services used in architecture, industries etc.
- IP Component (as per IP component framework) - defines that repository IP included supports IP components and therefore all the BOM items that component framework requires to be supported, e.g, component specification, SOW fragments
- Readiness - if repo supports readiness it includes a standardized way of linking to readiness materials for the IP being developed or readiness content being captured. For example, repo sources themselves could have more readiness content source material while lens can point to Readiness Board
- Offering - offering itself is not necessarily part of repository, but delivery IP in repository could support one or more offerings. This lense enables standard way to define the offerings that the delivery IP in repository supports
- Customer Questionnaire (to help scope) - some of the offerings in ACAI have leveraged structured questionnaires to get insights into the scope of upcoming engagement. This can be another example of lense that repository can commit to having to support sales of the IP being developed by the team.
- ... - there can be other types of assets - the goal of Evergreen Delivery IP is to implement support for the most common and allow also extended MCS teams to contribute back new lens content for specific IP types these teams are leveraging.

Becoming Evergreen

If you want to start implementing these principles into your IP work - teams and repositories, please see Getting Started in Guidance section.

Contribution Culture

Although previous topics of team model that supports contribution through innersource principles and diverse IP support are important to achieve broad spread of the approach and uniformity of delivery IP management supported through guidance and ready-made tools, arguably one of the main elements to sustain these uniform approaches long term is **contribution culture**.

Changing culture is journey that consists of multiple different approaches and activities, support contributing to overall change.

Evergreen delivery IP initiative's approach is to drive sustained IP management and support the overall cultural change through the mechanisms defined in subsections below.

Measurement based evolution of IP

Leveraging internal contribution measurements as well as IP impact internally and externally to showcase best contributions, IP value that should allow further investments both resource wise as well as focus wise.

See more on measurements

Insights at all levels

Using the measurements captured against the IP repositories providing insights at individual, IP team and organization level.

See more on insights at all levels

Contribution from day 1

Ensuring that contribution culture, innersource principles are front and center for all new hires into ACAI teams so they know about it, start contributing even in a small way and know that this is expected to grow over time.

See more on contribution from day 1

Innersource community

Community of practitioners of innersource (via evergreen techniques and tools) within MCS/ACAI that would be run by set of dedicated innersource PMs:

- organizing best practices exchange,
- rewarding/highlighting best contributors,
- highlighting top evergreen projects to have constant spotlight on benefits of contribution,
- driving the evolution of this Evergreen Delivery IP program forward based on ever-evolving needs and priorities.

See more on innersource community

Measurement Based Evolution

Key metrics identified as key for implementation of the evergreen approaches are defined in this section. Section covers external impact, internal impact metrics important at IP team level, and culture impact metrics that are more important at various organizational levels.

In addition to defining the levels of measurements envisioned and examples of the measures, this section also describes the approach for tracking and presenting measurement values.

IP External Impact

- Customers/projects using
- Geo-distribution – timezones/countries using
- Cloud revenue impact
- Delivery value impact
- Customer satisfaction in projects used

IP Internal Impact

- Repository stars
- Activity (contributors & contributions)
- Core team vs. external contributions
- Contributors by role/location
- Linked IP count
- Freshness: how often shows in queries made, used in SEE suggestions
- Time to proposal for opportunities linked to IP

Culture Impact

- Aggregated external and internal metrics across MCS IP repos
- Partner and customer co-innovation repos metrics
- IP fragmentation metrics (the same solution area – many copies)
- Contribution, participation and consumption

More on various levels of culture impact measurements and how they can drive culture change see section Insights at all levels.

Tracking & Presenting

Given there are various tools used for the underlying implementation of evergreen repos (Git and ADO), there is a separate Teams environment used as IP forum environment for discussions and each of the lenses ahs its specific approaches, and impact of the IP in MCS engagements can be received only from the tools tracking sales and delivery (C1 and Virtuoso), getting measure values as defined above for insights into team collaboration aspects and business impact requires integration of multiple data sources.

To simplify gaining the insights, at individual team level PowerBI service is used to aggregate various available data sources and present the KPI information. For measurements over multiple teams and to provide cultural insights at the level of organization, additional centralized data aggregation tools are needed. This means that there are 2 implementation levels of providing measurements:

- guidance and tools (initial sample dashboards and information on how to configure them) at the level of a single team and providing team level insights,
- centralized metric aggregation over all registered distributed repositories and tools to ensure end-to-end visibility with appropriate security.

For more details on how to set up tracking internal and external impact, see section Guidance.

Insights At Multiple Levels

Driving contribution culture depends on having insights not just at the team level, but also at the level of organization and entire ACAI. This enables employees to showcase their contributions, allows managers to focus their team on contribution work, enables higher management to make investment decisions and prioritization.

Evergreen Delivery IP goal is to provide insights at multiple levels so that individuals, their teams (including their managers) and leadership can have access to aggregated metrics across multiple IP projects, not just a single IP repository.

To achieve this, tools are needed to aggregate metrics from outside of the repositories, yet repositories should have these metrics being gathered.

Individual Level

Individual level insights contain information about contribution of a single contributor across multiple repositories they have used, reported issues, provided other feedback or contributed to by working on IP content.

These insights should support individual in their discussions with management as part of contributions review, also should be used to assign badges and promote best contributors by innersource PMs as part of the overall community of evergreen practitioners.

Team Level

At the team level managers get insights about their employee contributions and about overall contribution of the team. This can be a tool for both evaluation of the employees, but also for managing team targets of contribution to IP work allowing managers to prioritize some people more on IP work, some on pre-sales for example to balance overall team contribution to IP with other activities.

For a longer term impact on the overall culture of contribution these insights need to go hand-in-hand with having team targets updated to ensure culture of contribution is one of the important elements being tracked.

Organization Level

At organization level the insights are at the aggregation level of the entire ACAI team, for example, and allows leadership team to decide on the impact of IP being worked on, assess additional investment areas either in specific IP areas or overall culture support to ensure sustained focus on IP work.

Metrics of what IP is being worked on and how could be used at this level to track certain initiatives, for example, if there is a goal of increasing co-engineering with partners and customers, this could be tracked through analysis IP teams that are hybrid and involve also partner and customer contributions.

Contribution from Day 1

To have significant impact in contribution and to drive change towards more contributing culture pervasive in the organization, the practices and suggested behaviors need to be highlighted, promoted and enforced from day 1.

Therefore practices that other implementations of innersource approaches have used in other companies is to have contribution tools, expectations set up even during onboarding of all new resources. Of course these needs to be aligned also to goals of the teams and individuals and supported by leadership to achieved broader impact.

With this to ensure sustained contribution to drive evergreen delivery this is important to add the elements defined below to the onboarding of the employees into ACAI teams.

Guidance on Innersource

Basic training on innersource principles and various roles in the overall evergreen delivery process is required - what core team members should do (aka trusted contributors), what should be done and what are the responsibilities of contributors, what are the best practices/patterns of running successful community.

This goal requires training materials developed and included in onboarding and can be achieved by leveraging Innersource community.

Walk-through of Evergreen Approach

Basic training on principles, approaches of specifically evergreen delivery IP approaches that are implementation of the innersource principles in ACAI. This set of training materials needs is developed and maintained as part of evergreen delivery IP initiative and its delivery needs to be included in onboarding experience.

Request to Join the IP Team

Previous two points are about need, context and tools/resources that employees can use to start working on evergreen IP. But "practices makes perfect" thus, based on experience of other companies implementing innersource expectation/ask to have new employees start contribution to the IP that they are interested in and initially maybe do that by tackling "first issue" type of simpler issues is a great way to experience this approach in practice.

Innersource Community

Innersource community's main tasks are to be the engine behind:

- noticing, aggregating and sharing back best practices of doing evergreen delivery IP in ACAI back to other ACAI members,
- making sure evergreen delivery approach is evolving based on feedback and business goals,
- promoting contribution behaviors through showcases, badges and other gamification initiatives.

Innersource Program Managers

Driving evergreen delivery program forward and also ensuring there is a constant promotion of the contribution is a significant effort if we are intentional about it. To support it, there is a need for dedicated PMs (ideally representing all timezones) who would be the drivers of the evergreen program. This means either part time or even full time innersource PMs are needed.

Innersource PMs for the technical implementation and guidance would rely on evergreen guidance team maintaining the implementation details, aligning on the level of integration with other teams (e.g., BEO team). In this sense innersource PMs would be taking the role of the product owners for the "evergreen delivery IP implementation core team".

Innersource WW Community

Those participating in defining evergreen delivery IP approaches and guidance already have their own v-team, but to identity and promote best practices broader, an approach is to establish WW Community. The community co-leads are the same innersource PMs described earlier, yet the members should be representing all core team members working on IP so that they can exchange their experiences and learn about the latest tools and techniques available.

Use of community mechanisms that include elements concepts like SMEs, champs and regular community calls would be reusing the existing successful way of build out of a community of practice.

Getting Started

This section provides specific guidance on staring with evergreen delivery IP approaches, showcases how to set up the overall team structure and start adopting more and more lenses as a way of enriching the repository as it grows and needs more rich discovery. It can hopefully allow team adopting evergreen principles to avoid some of the blockers (aka 'evergivens') and ensure less effort needs to be invested to set up well working environment.

Core

First thing to start with making your IP work evergreen is to enable the repo to follow the structure and metadata guidance of the Evergreen Delivery IP. It is combination of aggregated best practices to promote innersource contribution model on one hand, it is specific technical/tools guidance also based on best practices gathered from various MCS teams.

This enablement included setting up or attributing the existing Git repos on Azure DevOps or on GitHub Enterprise, it also involves ensuring the correct metadata exists in the repository. And finally - it includes setting up Team site for all the contributors to have informal collaboration and meeting space as well.

For specific guidance on setting up Core evergreen elements for existing or new IP asset being created please see Core Structure Guidance.

Content Lenses

Based on teh description of the overall approach section (see Diverse Delivery IP) evergreen IP in addition to core element guidance provides guidance for multiple different types of IP being managed.

Each of these are called "content lenses" and provide more guidance on how specific type of content needs to be managed. If Core approach and metadata is a must for all evergreen teams, then the lenses implementation is optional and depends on maturity of the IP and type of IP being managed. Team can decide to use zero or more lenses for their repository. Using lenses make IP repositories more uniform and therefore provide lower barrier of entry to new contributors. For example, using document lens to provide documentation of APIs of the IP being created helps new contributors who have worked on documentation in other evergreen project to start contribution easier in a new project.

Lenses also act as extensibility model for the content management approach allowing over time new lenses being added with specific approaches of how they get managed. The list of currently supported lenses (and the lenses planned) is provided below with links to specific guidance on how to introduce the lens into your IP work.

If you want to contribute a new lens or request one that would be useful for multiple teams, please contribute your suggestions on Evergreen Delivery IP forum team. Note that even if you do not find the lens you need, you should enable your repo with new lenses as they get added.

List of currently implemented lenses with guidance:

- **Documentation lens** - enables teams to add structured documentation and enable documentation publishing site hosted for all MS employees.
- Work in progress: **Reference Architecture lens** - enables teams to document reference architectures in a structured way (leverages documentation lens for publishing the reference architecture).
- Work in progress: **IP Component lens** - enables structured management of IP component supporting materials of IP component framework
- Work in progress: **Readiness lens** - allows structured way of linking readiness materials for the IP from the site and publishing the readiness materials as Readiness Boards

If you want to present the approach, see this presentation covering the same aspects of the document lens approach and summary of all implemented lenses.

Describing the IP Assets

The approach for describing the core IP and also all the lenses in it is based on set of linked metadata files. The main metadata file and zero or more metadata files for specific lenses that IP repository is using.

Illustration of the approach is presented in the image below.

Core Structure

As documented in the Tools section of the Team Model page the hosting of core assets of the IP team is split over two main data sources:

- **Repository** based on Git and containing:
  - structured metadata used to index the IP content
  - IP sources themselves organized using pre-defined evergreen content lenses or just in a custom way
- **Teams site** used for collaboration and storage of binary assets (e.g., PowerPoint sites, Word documents)

Approach enables multiple external catalogue tools to index the repositories and ensure their discoverability on one hand, automatically push back feedback and issues to repositories on the other hand.

This section provides detailed guidance on how to:

1. set up repository (if one does not exist) or enable existing repository to comply with Evergreen Delivery IP guidance
2. set up collaboration site based on Teams for the IP core team and contributors as per Evergreen IP guidance
3. register the set up hosting sites with centralized search tools like Chrysalis, SEE that ensure discovery and feedback integration

Information on lenses supported and guidance on setting them up is provided separately (see list of lenses in Getting started page).

Repository

As described earlier, the Evergreen Delivery IP approach enables teams to achieve both:

- "light up" their current repositories to start following Evergreen Delivery IP principles and already starting to benefit from integrations enabled through adding minimal set of metadata and registering the repository with the suggest indexing tools
- create a new Git repo in ADO or GitHub Enterprise environment of Microsoft and add the necessary minimal set of files to enable Evergreen Delivery IP compliance.

Specific guidance on setting up repository is provided in Repository page.

Specific guidance on setting up the core metadata content is provided in these pages:

- readme.md
- contributing.md
- evergreen.yml

Collaboration Site

Collaboration site is based on MS Teams platform and is primarily used for team's informal communication, having and recording sync meetings. It can also be used to store binary content, e.g., PowerPoint presentations and Word documents.

Despite being capable of storing a lot of IP content, to drive structure and evergreen IP principles, it is suggested avoid documenting the IP in binary formats like Word as primary formats where possible, but rather to use repo based documentation published using documentation lens guidance. Binary content required can be linked from a structured documentation site via Teams (SharePoint) links or through embedding the content.

Collaboration site has these properties:

- created as open site for all MS employees to freely joined
- has documentation as part of description
- allows for Core team private conversations through a (set of) separate private channel(s)

More on the suggested settings and structure of the site to enable innersource based communication, please see Collaboration Site page.

Register with MCS Tools

Tha main thing to be done to enable indexing and discovery of the repositories and Teams sites with the global MCS tools used in pursuits and deliveries is to register the IP repositories with project Chrysalis. Chrysalis implementation is in progress, for the latest information see: https://chrysalis-innersource.azurewebsites.net/.

Evergreen Delivery IP team is working with project Chrysalis and knowledge management team (SEE/Campus) to have automated registration/crawling of Evergreen Delivery IP sites if they get registered in well known repository locations.

Example

Example of the team following this approach is Evergreen Delivery guidance IP team itself having:

- Git based ADO repository with necessary metadata files and other data specific to the project
- MS Teams team used for collaboration with broader community and enabling private communication of the core team.

Repository

This page describes the guidance on enabling existing repository or creating a new repository for capturing the IP in a structured way.

Types Supported

Given a long history of MCS teams using ADO and the upcoming focus on GitHub, the Evergreen Delivery IP approach supports Git repositories hosted in both these platforms:

- ADO repository hosted as part of any of the Microsoft organizations.
- Git repository hosted in GitHub Enterprise MCS collection (limited access to this for users in Microsoft on-boarded on GHAE).

Note

Note that GitHub Enterprise MCS collection is currently being rolled out and has limited repository onboarding supported until full processes around security/scanning etc. get established. The intention is that over time it should become the primary location for IP work and also project delivery work as also more ADO features are migrated to GitHub Enterprise.

Provisioning a Repository

ADO

It is suggested to use one of the well defined and managed organizations in ADO for storing services IP assets, for example:

- https://servicescode.visualstudio.com/
- https://servicesdocs.visualstudio.com/

To create an ADO repository for managing evergreen IP, please:

1. proceed to Services DevOps ServicesCode Repo Request page,
2. fill out the request details and submit the request for repo provisioning, when filling out, choose Internal Project Type.

Note

Take into account that Security groups need to be set up that will contain list of administrators. Do that before submitting the request on http://idweb/. For more informaiton see Services DevOps Wiki.

In addition to these predefined well-known and managed locations Evergreen Delivery IP supports ADO repositories hosted in other organizations of Microsoft. To use them, add the repository to one of the organizations. Note that custom organization will not have the same support by Services DevOps teams, might lack some of the useful tool integration, e.g., SonarQube.

Git Hub Enterprise

Onboarding for GitHub Enterprise for MCS employees currently is being piloted and planned to be rolled out broadly in H1 of FY22. If due to specifics of the project there is a need to use GitHub Enterprise repository, please submit and AdHoc request.

Note

Note that pilot onboarding is limited and there is no guarantee that GitHub Enterprise access will be granted based on your request.

Mandatory Metadata

After provisioning the repository, please make sure to add these three files to the root of it to enable it for innersource as per Evergreen Delivery IP approach:

- readme.md
- contributing.md
- evergreen.yml

Repos per IP Asset

There IP assets that are large and require multiple repositories. For example, there can be a microservices application where each service is being managed in its own repository, yet all of them are manged by a single IP team.

In these cases when IP assets have multiple repositories, all of them would need to follow this guidance and require Evergreen metadata information added that would link them together as representation of a single IP asset. Also all of them would be linked to the same collaboration site for the IP team.

Example

- Git based ADO repository of the Evergreen Delivery IP initiative
- Git based GitHub Enterprise repository of DevOps DoJo innersource project

Collaboration Site

An important aspect of any IP team is collaboration that goes beyond just pull request and issue information tracked in teh repositories. There is a need for space for more informal discussions, for binary documentation that is not the best for structured source control environments like Git and for supporting remote meetings. This is achieved through IP team Team sites.

The general approach for Teams site reuse the learnings and approaches from experience of teams running IP Forums and DevOps DoJo. As described in the Approach Section under Team Model, there is a single site per the logical IP asset being worked on by a single Core contributor team. Note that if IP Asset is quite large, e.g., microservices based application that is built up of multiple services each residing in its own Git repository, then this single collaboration site is linked from multiple repos.

Naming the Collaboration Space

Name of the Teams team that is used as the collaboration site must represent the IP asset being worked on. If this is asset is an "e-invoicing solution for tax authorities" then "e-invoicing", "e-invoicing solution" would be good examples of the names to be chosen.

Team Owners

Core Team members need to be made owners of the Team site. As per Microsoft policies, this means that each IP team consists of at least 2 core team members.

Access Rights

Given Evergreen follows innersource principles, by default it is suggested that the Team is set up with public access for all Microsoft employees. This means that general channel and other default channels added would be visible to all users who want to join and learn or contribute.

To ensure Core team (trusted contributors) have their own private space for discussions, Core Team private channel with access granted to the Core team members can be created. All files, OneNote notes and other assets in this protected channel would be visible only to core team members.

Note

The only downside of this approach is that Teams currently does not allow meetings scheduled inside of the Teams private channels. If meetings cannot be kept open for entire community to join, suggestion is to use Meet Now or schedule recurring Teams calls outside of the channel. We hope this feature gets added in Teams soon to remove even this annoyance (aka 'evergiven').

Linking Other Content Into Teams Site

To ensure efficient visibility into the teams work and work results, metrics, it is suggested that Teams site general channel and/or core team's private channel has additional tabs added, for example, to include:

- **team dashboard from ADO** showing the current work in progress or current issues list directly in Teams environment
- **documentation lens** web site if it is part of the IP asset
- **metrics dashboard** showing important internal and external metrics team is tracking

Reusing Content

The binary content like PowerPoint files can be hosted in the documentation web site through embedding via shared embed link. This enables common use of content from structured repositories of the IP and the collaboration site. When sharing the content from collaboration site via embedding, please ensure that this is shared from the public part of the site available to all MS employees.

This is an example of Evergreen Delivery IP overview deck exposed from the IP collaboration site:

Example

MS Teams team of the Evergreen Delivery IP guidance used for collaboration with broader community and enabling private communication of the core team.

Example of the structure of a simple collaboration site is shown in the screenshot below. It contains general channel for communication with broader community, dedicated public presentations channel where presentations content gets stored and protected channel for the core team to communicate.

Note that team can decide to create new public or private channels as per needs of the team to ensure structured and well managed collaboration environment.

readme.md Content

This page provides guidance about the approach and suggested structure of defining the **README.md** for the project represented by an evergreen repository.

Introduction

Readme file is one of the first files that would need to be created as part of any Evergreen Delivery IP project being started to capture the main information about it, provide useful pointers to additional elements in the repository or the project team site.

Although in general there is no hard rules around the format and content of the readme file, as a general guidance and given the tools promoted by the Evergreen Delivery IP it is suggested to use:

- Markdown format for creation the file (readme.md)
- Follow the general content structure described below.

Note

Note that given IP repositories can be just basic repositories without having additional lenses and they can be repositories with enabled self-hosted or centrally hosted documentation site, the content location of the readme.md content might be different. General rule is that readme.md would mostly reference other content in external documentation of the project if one exists and it would contain more description in readme.md if there is no external documentation site and contributors would see more the files directly in repositories (ADO and GitHub).

Suggested Structure

The structure of the readme file is suggested to include the sections listed below.

Name

Name of the project or IP that is managed within the asset must be provided. This is the name that people will refer to when talking about the project or IP being created.

Note

If the project is not active anymore and there is no active contribution it is suggested to include note about this status as early as possible in the readme file.

Description

Description of the project must provide more details about the project itself, its goals, challenges it is solving. If the IP is a solution/component/script, describe what this component or script can do. If this is a guidance IP, described what this guidance is about, what benefits will it provide to those following this guidance.

Make sure to include who this IP is for - might be based on role, might be based on type of activity or even might be restricted to a certain geography or industry.

Getting Started

In addition to overall description provided in previous section, it is suggested to include a short list of actions that can be taken by the target audience to learn more about the initiative or even to start a discussion and plan contribution. This ensures there are "affordances" included that help to point people on how to get more information or start contributing.

Installation/Usage

If the IP produces material that is something that can be installed or used otherwise.

For example:

- a tool that needs to be installed on client machine to use it,
- guidance documentation that is being published from this repo into external documentation site),
- a library that has API exposed and is installed as a NUGET package to include it into someone else overall solution.

Use this section to describe how the IP produced out of this repository can be installed and used. In simple cases this could be a link to a web site that, for example, is built using code in this repo. In more complex cases where the repository contains a microservice of a large solution there can be installation guidance and reference to overall solution description.

Support

Describe how users of the IP can provide feedback, get support if they face any issues. Ideally the team should include communication channels that can be used to request support. Could be a channel in Teams and/or list of issues that users can submit their issues in and track their status.

Contribute

Evergreen Delivery IP compliance repositories are following innersource principles. Thus and important aspect of the repository is contribution rules and approaches. This section must provide introduction into contribution approach and link to contributing.md that should expand on it in details.

Core Team

In any innersource project most of the contribution comes from the trusted contributors (called here *core team*). Readme should have a list of these trusted contributors so it is clear for other contributors who, for example, to reach out in case of certain questions etc.

If possible and applicable, please include information on the focus area of a contributor of IP set is large and there are different hats that each of the core team's contributors have.

Acknowledgements

Acknowledgements section can include reference to contributors (even if they are not part of the core team), it can also include lists of other repositories and components that are used as part of the IP being managed in the repo.

Scope

Although innersource by default is opened to entire organization, there might be cases where initially repositories and teams are more restricted. Also is possible that certain repositories are opened for co-innovation work to Microsoft partners and customer representatives. This section must provide information on the "scope of visibility" of the IP work so contributors have full clarity on this.

Example

Example of the readme.md file of the Evergreen Delivery IP guidance project itself is available in its repository.

contributing.md Content

This page provides guidance about the approach and suggested structure of defining the **CONTRIBUTING.md** for the project represented by an evergreen repository.

Introduction

Contribution file expends on the content of the readme.md by focusing specifically on the contribution aspects of the project. It must address three groups of the stakeholders doing "different types" of contributions:

- **project core team members** (trusted contributors) who are responsible not just for contributing but also welcoming contributions of and supporting other contributors and consumers
- **project contributors** who can provide different types of contributions depending on the type/depth of the IP - starting with updates to documentation and ending with contributions to the main IP elements of the solution managed in the repository
- **project consumers** who are using the *"product"* of the IP repository and can report back issues and also feature suggestions, additional feedback that can drive the priorities for the core team.

Although in general there is no hard rules around the format and content of the contribution file, as a general guidance and given the tools promoted by the Evergreen Delivery IP it is suggested to use:

- Markdown format for creation the file (contributing.md)
- Follow the general content structure described below.

Suggested Structure

The structure of the contribution guidance file is suggested to include the sections listed below.

Content Structure

Content structure must describe all the locations where important content of the overall project gets stored. Primarily it should focus on describing the structure of the files in the Teams file store and the structure of the repository that contains structured IP content.

Contribution Process

Contribution process needs to provide the process details around providing different types of contributions by different roles within the overall project. This includes feedback and issues, this can include informal feedback during supported informal communication channels (e-mail, Teams calls), this must also include pull request based contribution of changes to the structured content.

Section must also describe the tracking approach for changes being made. Ideally all structured contributions and even changes in binary information storage as part of the Teams site must happen as a result of the issue being addressed in repository.

Tools

This section should include information for the contributors on the types of tools that are used for managing the content, developing and building it, managing the contribution workflow, communicating, tracking important team metrics.

In addition to just listing tools it is important to include links to externally published materials or even additional information in repository describing how to deploy and configure these tools so the initial barrier of contribution is removed as much as possible.

Example

Example of the contributing.md file of the Evergreen Delivery IP guidance project itself is available in its repository.

evergreen.yml Content

evergreen.yml is used to provide the main metadata description of the IP assets repository contains, links to team site and the core team working on the IP. It also includes information on additional content lenses that repository contains so that they can also be discovered. It is the tool to "light" up existing repositories with additional metadata that external tools over time can pick up, ensure more discoverability and integration of feedback.

Data Model

Data model for the evergreen.yml file is a **single** object of type **Evergreen IP Metadata Object** containing:

- information on the version of the metadata that the IP supports
- core IP information like name and description, image URL
- information on core team (trusted contributors)
- links to:
  - IP Forum team site where team collaborates, has meetings
  - repository where issues (backlog) of the IP are managed and where new issues/requests can be submitted
- tags and taxonomy values so that this information can be used in more structured search for IP
- list of lenses that the IP commits to and maintains within the repository

Example:

```
- version: 0.1
  id: evergreenip
  name: Evergreen Delivery IP
  description:
    URL: https://aka.ms/evergreenip
    text: Evergreen Delivery IP is ACAI CTO driven program to improve the way CTO and field teams are finding, reusing, becoming contributors and leading evolution of managed and community IP. This is repository containing the guidance and supp
  created: 2021-03-24
  core-team:
    - name: Name Surname
      email: name@surname.com
      URL: https://aka.ms/evergreenip
  collaboration: https://URL-to-collaboration-space-on-Teams
  issues: https://URL-to-the-repository-backlog-issues-list
  tags: IP,ACAI,ACAI CTO,guidance,pattern
  taxonomies:
    - type: product
      value: Azure,Dynamics,not product specific
    - type: industry
      value: civilian government
    - type: language
      value: en
  lenses:
    - type: docs
      metadata: evergreen-docs.yml
    - type: ipcomponents
      metadata: evergreen-ipcomponents.yml
```

Evergreen IP Metadata Object

Property Name	Required or Optional	Type	Description
version	required	string	Version of the evergreen IP supported. Must be 0.1 currently
id	required	string	Short name (no spaces allowed) that must be unique identifier of the IP asset
name	required	string	Name of the IP asset/repo
description.URL	optional	URL	Description of the IP Asset if there is a full web site that supports it where users can get more information
description.text	optional	string	Short text based description of the IP asset so it can be used to display as a summary on the search page, for example.
created	required	date	Date when the IP repo was created and team established.
core-team	required	array of **Team Member**	Contains information on the team members.
collaboration	required	URL	URL to the MS Teams collaboration site used by the team on top of structured collaboration in the repository.
issues	required	URL	URL to the Git repository tools on ADO or GIT where users can view and submit issues/requests, understand the backlog of the IP initiative.
tags	optional	string	List of comma-separate tags that describe the initiative (not from a managed taxonomy).
taxonomies	optional	array of **Taxonomy Value**	Optional list of all the taxonomies supported with taxonomy values. Each taxonomy in the list must be present not more than 1 time.
lenses	optional	array of **Lens Information**	Optional list of all the content lenses supported by IP asset. Each lens in the list must be present not more than 1 time.

Team Member Object

Property Name	Required or Optional	Type	Description
name	required	string	Mandatory name and surname of the team member
email	optional	email	E-mail address of the team member.
URL	optional	URL	URL to the web site that contains information on the team member.

Taxonomy Value

Property Name	Required or Optional	Type	Description
type	required	string	Unique name of the taxonomy. Currently supported: product, industry, language
value	required	string	Comma separate list of mapped taxonomy values.

Given taxonomies can have multiple levels, the guidance on choosing values for the taxonomies are:

- **product** taxonomy - need to specify "Leaf node default label" values from the Product taxonomy definition file
- **industries** taxonomy - need to specify "Leaf node default label" values from the Vertical industries taxonomy definition file
- **language** taxonomy - need to specify ISO 2 letter code for the language as per ISO 639-1:2002

Lens Information

Property Name	Required or Optional	Type	Description
type	required	string	Unique name of the lens supported. Currently supported: docs
metadata	required	string	Path in the repository to the location of the metadata file. If relative path, then mapped from location of the evergreen.yml file.

For more content on the metadata file of the docs lens see Documentation Lens guidance.

Example

Example of a evergreen.yml file of the Evergreen Delivery IP initiative you can view in the root of the repository.

Documentation Lens

Documentation content lens supports the most IP assets development as almost any IP project requires documentation either to to have a description of the solution or description of its APIs or, for example, as description of the methodology or process etc. Documentation lens can also be used as a way to provide a nice web based documentation about IP asset, its management, contribution guidance so innersource contributors have easy way to onboard and start contributing.

Because documentation lens is broadly used, other lenses can also require this lens to be present to be able to publish their specific information as well as part of the IP asset documentation web site.

Content Tools

Given overall evergreen approach is promoting use of structured IP handling in Git repositories in ADO or GitHub Enterprise, the documentation is also:

- managed as Markdown based files with additional supporting files (images, embedded PowerPoint as needed)
- stored in Git repository,
- updated through pull requests to the repository,
- converted into published version using DOCFX static site generation engine leveraged my broader Microsoft to publish documentation as well.

Editing documents in markdown and leveraging Git pull request approach can feel taxing at first, but overall it ensures well structured and validated content with full tracking of history including structured deployment of the changes. To ensure it is more automated and easier, Evergreen approach suggests leveraging Visual Studio Code as the authoring tool.

Example of VS Code being used to edit this page is shown in the image below. Suggested plug-ins to enable easy VS Code authoring of documents are:

- Code Spell Checker extension
- Markdown All in One extension
- MarkdownLint extension

Content Structure and Metadata

Each content lens needs to have its metadata file that provides additional content type specific information that indexing tools can capture and use in linking to other assets, more detailed processing etc.

Documentation lens requires evergreen-docs.yml metadata file that is linked from the main metadata file of the evergreen repo. The metadata file contains main the details about the documentation managed in repo, points to the publishing site location.

In addition to metadata file there are other requirements and suggestions around the structure of DOXFX document content site. For more guidance on best practices authoring content as per DOCFX guidance see Document Authoring using DOCFX.

Documentation Web Site

To ensure documentation can be presented in an easy to consume way though and in a unified way (without differences when teams use ADO with its Wikis vs GitHub with its GitHub pages), approach promotes publishing the documentation on an external web site that gets automatically generated from the documentation source file, content structure guidance files, template override files as per DOCFX stored in the Git repository. The web site is secured to ensure only access by Microsoft employees.

Hosting of the generated web site is planned to be possible in two ways:

- **self-hosted site** where team hosts it in their own Azure subscription,
- **centrally hosted site** where centralized tools that index the evergreen sites see all sites with Documentation lens in them and on a regular basis compile and host the web site centrally.

Self-hosted Site

The process of generating web site and publishing it by the team itself leverages:

- Azure Pipelines to deploy the documentation and underlying web infrastructure
- Azure Web Apps to host the web site that is integrated with Microsoft Azure AD tenant to control user access

The approach provides the most flexibility of deployment, use of any custom website name, leveraging all features of Azure Web Sites that team can configure as they are responsible for it. On the other hand this approach requires own subscription to pay for

the hosted web site and configuration of CI/CD templates as per guidance on this site (should take approximately 2 hours to set it up).

This approach of hosting the site is fully supported and there are example pipelines for infrastructure provisioning and site deployment that can be used as examples to start implement the hosting process.

See more and get ready-made templates to help with implementing self-hosted documentation site in Self-hosted Site page.

Centrally Hosted Site

Benefit of centrally hosted documentation site is that IP team needs only to document the site using DOCFX guidance, ensure it is properly described in evergreen-docs.yml and external documentation publisher tool on a regular basis (or based on trigger) gets the updated content and publishes it.

The approach is more restricted from the point of view of the flexibility, yet it does not require own additional configuration of deployment pipelines, web sites and does not require own Azure Subscription.

Given this approach requires build-out of the centralized infrastructure to support it, this hosting option is currently not yet available (work in progress).

Example

Example of a web site maintained using described approach of documentation lens is this web site.

evergreen-docs.yml Content

This page describes the contents of hte Evergreen Delivery IP document lens metadata. It is used to ensure documentation stored as part of the repo can be indexed or processed by the external tools. Note that to simplify the overall approach some of the metadata values have default values allowing them to be omitted as part of the metadata file.

Data Model

Data model for the evergreen-docs.yml is a **single** object of type **Document Lens Metadata Object**

- lens: docs
- metadata version (current version 0.1)
- documentation source location (default /docs)
- location of the published documentation
- doc-generator overrides (optional because default assumes docfx being used):
    - type of the static site generator engine used (the only supported version is docfx); can be used in future to specify alternative tools for static site generation
    - location of hte main configuration file of the static site generator (default /docfx.json)

Example:

```
- lens: docs
  version: 0.1
  source: /docs
  published: http://aka.ms/evergreenip
  doc-generator-type: docfx
  doc-generator-config: /docfx.json
```

Document Lens Metadata Object

Property Name	Required or Optional	Type	Description
lens	required	string	Must be docs
version	required	string	Must be 0.1 currently
source	optional	string	Absolute path to location of the source of the documentation. Default value (if not supplied) is /docs
published	optional	url	Valid global URL pointing to the location where the documentation is published. If not supplied, it is assumed that documentation is not being published.
doc-generator-type	optional	string	Defines type of static site generator used. Must be docfx in current implementation. Default value docfx
doc-generator-config	optional	string	Defines where the main configuration file of generator is stored (used by remote generator). Default value /docfx.json

Document Authoring Guidance

This pages contains additional guidance on managing the structure of DOCFX compliant documentation site, tips/tricks that might be useful for those maintaining the documentation.

Source Documentation Structure

In addition to main document lens metadata file evergreen-docs.yml DOCFX requires additional files and certain document structure to be used. The main elements to pay attention to when implementing own documentation site are:

- all the content of the site is stored under /docs (can be changed, but this is the suggested location)
- DOCFX main configuration file used to build the static site stored in /docfx.json (see docfx.json example)
- have optional /docfx/templates/<yourtemplatename> folder where files overriding the template are located (if template is overridden)

The typical structure of the documentation section of the IP asset would typically have this guidance:

```
|-- / (root)
    |-- docs
    |     |-- index.md
    |     |-- <other page>.md
    |     |-- toc.yml
    |     |-- <subfolder>
    |     |      |-- index.md
    |     |      |-- <other page>.md
    |     |      |-- toc.yml
    |-- docfx/templates/<my project name>
    |     |-- favicon.ico
    |     |-- logo.svg
    |-- docfx.json
    |-- evergreen-docs.yml
```

Generating Site Locally

When working with documentation it is useful to generate it locally, validate via http://localhost:8080 that is the default hosting environment for local documentation site. To be able to process the source files into static site, docfx command line tool needs to be downloaded, unzipped and added to PATH.

The command line docfx .\docfx.json -o "_docfxsite" –serve is used to generate the site and server in on localhost:8080.

To ensure that the generated site HTML and other content files are not checked back into the source control, please update your .gitignore to include the _docfxsite location.

More information on using DOCFX tools: DocFX command-line, Docfx commands.

Site Structure and Menus

Top menu and side menu for the site can be created using toc.yml files as part of the overall documentation content definition. More information on the TOC file contents you can find in the DOCFX documentation on TOC creation.

Examples of the TOC files:

- for this site covering main menu,
- for this site's one of hte second level menus.

Customizing Look and Feel

Probably the simples way to customize look and feel of the documentation site generated using DOCFX is to export the default template, find the files to be modifies and replace them in a custom location that is referenced from docfx.json file.

More details on exporting default template, defining new location for the override files are provided in DOCFX tutorials.

Example of one of a minimal update is shown as part of this site where only favicon and main site icon are replaced with customized versions.

Embedding Resources

Markdown allows using links to link external resources hosted on the web. For binary content like YouTube Videos, PowerPoint presentations, for example, it is suggested to host it as part of the web site in embedded format.

Internal Video

Embedding a video about the IP, its use or contribution approaches is suggested through the Microsoft Steam. The video can be uploaded under the IP team group's videos collection an shared out.

Image below shows the metadata being entered during upload of the video to Microsoft Stream and embed link being retrieved to use it in documentation website.

The `div` content copied from the sharing as embedded screen needs to be added directly into the markdown. See the guidance page for deploying the site showing how embedding of internal video works. It is suggested to extract the embedding link as responsive so it fills the entire width of the content area.

Documents

Embedding Word or PowerPoint requires opening the PowerPoint or Word document in On-line version of the app, choosing Share -> Embed and copying the embed `iframe` code to the markdown file directly. Image below shows Word UI used to request embedding link.

Following image shows link embedded directly into DOCFX markdown content file.

External Video

For more information about embedding external YouTube video see DOCFX Favored Markdown description.

Self-hosted Documentation Site

Page provides information on the approach of implementing the Self-hosted Documentation Site. Self-hosted documentation hosting is one of the options of hosting as per the documentation lens approach defined.

Self-hosting provides the most flexibility and control, yet requires to use own Azure Subscription to host the documentation website and also needs some more work to set up the pipelines for continuous integration and continuous release of the documentation updates to the web.

There are two elements required for self-hosted documentation sites:

- web server infrastructure integrated with Azure AD of Microsoft hosting the documentation web site, supporting AppInsights as well
- automatic build and deployment of the DOCFX based site using the source files in the repo and pushing them to the web server infrastructure.

The sections below and child sections provide guidance on how to set up Automated Builds (Continuous Integration) and Automated Deployments (Continuous Delivery) for these two elements.

Branching strategy

Distributed version control systems like Git give you flexibility in how you use version control to share and manage code. A branching strategy should find a balance between this flexibility and the need to **collaborate** and **share code** in a consistent manner.

Adopting an effective branching strategy will help to smoothly collaborate and integrate work frequently to keep the greatest latest content continuously published.

To avoid long-lived branches, we adopt a Trunk-based development branching strategy, in which any new change will be proposed in a feature branch that ideally will not live longer than 24 hours (when individuals on a team are committing their changes to the main branch multiple times a day, it becomes easy to satisfy the core requirement of Continuous Integration and Continuous Delivery).

Branch name / naming convention	Purpose	Example
main	The default branch, containing the code running in the production environment. Previously, the default branch was almost always named master (for more background on this change, this statement from the Software Freedom Conservancy is an excellent place to look).	
[user alias]/[description-of-change]	The feature branch, also known as topic branches. Feature branches isolate work in progress from the completed work in the main branch. Making it short-living will help achieve a faster integration of the changes, a faster review and approval and thus more transparency	

Environments strategy

As with any software, we need a strategy to validate our changes before releasing those changes to production. Even if we have tools to generate the website locally, the hosting infrastructure of our site will be different, and before releasing the new changes, we might want to add automated tests to every change (e.g. grammar check, length check, accessibility check) and see the results before releasing to production.

Following the "build once" strategy, the same automated build artifact must be deployed to all environments.

As environments strategy, we recommend having the following environments:

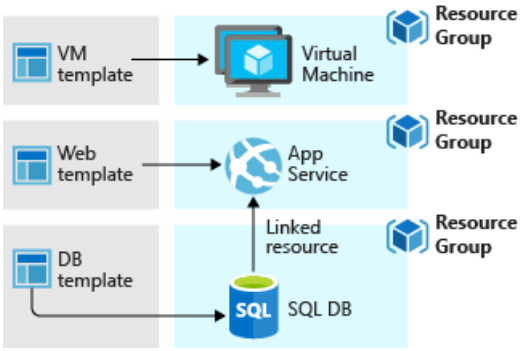
Name	Purpose	Trigger conditions	Example
Integration	This environment will allow any contributor to see the changes being worked in the contributor branch before event starting a new Pull Request.	Changes to any branch different from main will trigger a deployment to this environment	https://evergreenip-int.azurewebsites.net/
Staging	This environment will allow any approver to see how the site will look in production before approving the Pull Request	Any Pull Request to the main branch will trigger a deployment to this environment	https://evergreenip-staging.azurewebsites.net/

Name	Purpose	Trigger conditions	Example
Production	The environment with the approved content publicly - or internally - available to anyone	Changes to the main branch will trigger a deployment to this environment	https://evergreenip-staging.azurewebsites.net/

Infrastructure as Code

To speed up configuration, deployment, maintenance and evolution of the hosting infrastructure, avoid environment configuration drift, and achieve the full discipline of Continuos Delivery, we will manage infrastructure as code leveraging MCS infra-as-code-source assets, principles and practices, in concrete:

1. Pipeline orchestrated deployment
2. Module approach for Infrastructure as Code



See guidance on how to structure the infrastructure as code for all the environments of the self-hosted site in the child article Infrastructure as Code

Continuos Integration and Continuous Delivery

Continuous Integration and Continuous Delivery for both the site content and the site infrastructure will help bring the following benefits:

1. Enable asynchronous collaboration, parallel development and integration of the work frequently, and verified by an automated build (including test) to detect integration errors as quickly as possible.
2. Publish new updates or features in short cycles, quickly, reliably and at any time, following a repeatable and sustainable process

There are different tools that support implementation of Continuous Integration and Continuous Delivery practices:

1. See guidance on how to implement Continuous Integration and Continuous Delivery leveraging Azure DevOps Graphic User Interface pipelines in the child article Azure DevOps GUI pipelines
2. See guidance on how to implement Continuous Integration and Continuous Delivery leveraging Azure DevOps YAML pipelines in the child article Azure DevOps YAML pipelines

Pre-requisites

Azure DevOps project

Coming soon...

Azure subscription

Coming soon...

Deployment Service Principal

To implement CICD, we will use an Azure Service Principal.

Note

Follow instructions in article How to: Use the portal to create an Azure AD application and service principal that can access resources to create the AD Application to use for App Service authentication

The Service Principal needs to have the following configuration:

Setting	Value	Description Example
Certificates & Secrets	A secret needs to be configured to allow Azure DevOps deployment pipelines authenticate with Service Principal Credentials	

The Service Principal needs to be granted with the following permissions:

- Contributor in the Resource Group or Resource Groups that will used for the hosting infrastructure.
- Create a Service Connection in Azure DevOps

Infrastructure as Code

We implement Infrastructure as Code to speed up configuration, deployment, maintenance and evolution of the hosting infrastructure, avoid environment configuration drift, and achieve the full discipline of Continuos Delivery for Azure hosting infrastructure.

We leverage ARM Template and MCS infra-as-code-source assets, principles and practices to implement Infrastructure as Code consistently.

Strategy

Modular strategy

Note

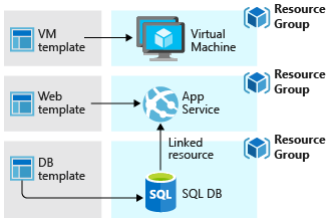
Learn more at MCS Infrastructure as Code Source (IaCS)

How you design infrastructure as code templates is entirely up to you and how you want to manage your solution. You can deploy your complex infrastructure through a single template to a single resource group; but development of infrastructure as code and configuration as code can easily become complex and unmaintainable, then it makes sense to divide your deployment requirements into a set of **targeted, purpose-specific templates**.

For infrastructure as code, same principles as in software development can be adapted and followed to help developers produce flexible and maintainable code (SOLID, 12 factor app, etc.).

> Following the SOLID principle of single responsibility we develop each module within its' own lifecycle and all the DevOps practices apply. Each IaC module should be reusable and have a single responsibility and thus only a single reason to change

A module is a **reusable combination of infrastructure and configuration as code templates and scripts that will let you manage your infrastructure in Azure through one or more idempotent operations (create, remove, etc.)**. Each module will target a single Azure Resource Provider.



Modules are written in an quite flexible way, therefore you don't need to modify them from project to project, as the aim is to cover most of the functionality that a given resource type can provide, in a way that you can interact with any module just by sending the required parameters to it – i.e. you don't have to know how the template of the particular module works inside, just take a look at the readme.md file of the given module to consume it.

The modules are multi-purpose, therefore contain a lot of dynamic expressions (functions, variables, etc.), so there's no need to maintain multiple instances for different use cases. E.g. there's only one VM template that covers all scenarios.

Pipeline orchestrated deployments

Note

Learn more at MCS Infrastructure as Code Source (IaCS) Deployment options

With pipeline orchestrated deployments, we use Azure DevOps pipelines as glue or orchestration for deploying Modules in an Azure environment.

Pipeline orchestration has two possible strategies:

- **Service-chains** (directly references the source code of Modules) - quicker and simpler to set up and use.
- **Uses Azure DevOps Artifacts** (Universal packages) - this is safer, as the solutions deployed in Azure are not directly associated with the source code of Modules

For simplicity, we use the Service-chains approach.

See Deployment guidance in sections:

- Azure DevOps GUI pipelines for graphical interface pipelines
- Azure DevOps YAML pipelines for YAML pipelines

Repository Structure

To enable the strategies mentioned in the sections above, we organize the repository in the following way:

```
|– / (root)
  |– pipelines/
  |   |– iacs/
  |   |  |– app-service-plan-windows.json
  |   |  |– app-service-windows.json
  |   |  |– application-insights.json
  |   |  |– key-vault.json
  |   |– infra/
  |   |  |– deploy.yml
  |   |  |– variables.yml
  |– platform/
      |– common/
      |  |– egipai.json
      |  |– egipkv.json
      |  |– egipsp.json
      |– int/
      |  |– evergreenip-int.json
      |– prod/
      |  |– evergreenip.json
      |– staging/
         |– evergreenip-staging.json
```

Where:

- pipelines/iacs/ directory contains all Infrastructure as Code modules that we will use to deploy our infrastructure.
- pipelines/infra/ directory contains the YAML pipeline and YAML variables file to implement continuous integration and continuous delivery of the infrastructure. See Infrastructure Pipeline
- platform/commons/ directory contains the configuration of the modules that are part of the common environment
- platform/int/ directory contains the configuration files of the modules that are part of the integration environment
- platform/prod/ directory contains the configuration files of the modules that are part of the production environment
- platform/staging/ directory contains the configuration files of the modules that are part of the staging environment

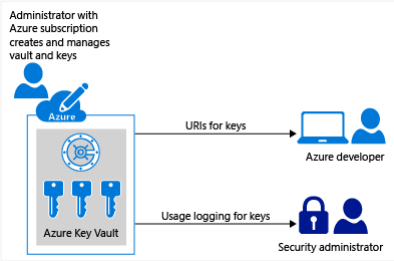
IaC Modules

Key Vault

Note

See Key Vault module in IaCs library

Azure Key Vault is a cloud service for securely storing and accessing secrets. A secret is anything that you want to tightly control access to, such as API keys, passwords, certificates, or cryptographic keys. Key Vault service supports two types of containers: vaults and managed hardware security module(HSM) pools. Vaults support storing software and HSM-backed keys, secrets, and certificates. Managed HSM pools only support HSM-backed keys



We use Key Vault to re-use unpredictable configurations such as Application Insights Instrumentation key required during module deployment.

Note

See the Application Insights module that we use in our Evergreen Repository

```
{
  "$schema": "https://schema.management.azure.com/schemas/2019-04-01/deploymentTemplate.json#",
  "contentVersion": "1.0.0.0",
  "parameters": {
    "KeyVaultName": {
      "type": "string",
      "metadata": {
        "description": "Name of the Key Vault resource. Unique Public DNS Name, so no duplicated names can exist"
      }
    },
    "KeyVaultLocation": {
      "type": "string",
      "defaultValue": "[resourceGroup().location]",
      "metadata": {
        "description": "The supported Azure Region where the key vault should be created"
      }
    },
    "KeyVaultSKUName": {
      "type": "string",
      "allowedValues": [
        "standard",
        "premium"
      ],
      "metadata": {
        "description": "Azure Key Vault is offered in two service tiers, standard or premium"
      }
    },
    "KeyVaultEnabledForDeployment": {
      "type": "bool",
      "defaultValue": true,
      "metadata": {
        "description": "In order for Key Vault to be used with Azure Resource Manager virtual machines, the EnabledForDeployment property on Key Vault must be set to true"
      }
    },
    "KeyVaultEnabledForDiskEncryption": {
      "type": "bool",
      "defaultValue": true,
      "metadata": {
        "description": "For enhanced virtual machine (VM) security and compliance, virtual disks in Azure can be encrypted"
      }
    },
    "KeyVaultEnabledForTemplateDeployment": {
      "defaultValue": true,
      "type": "bool",
      "metadata": {
        "description": "When you need to pass a secure value (like a password) as a parameter during deployment. The value is never exposed because you only reference its key vault ID"
      }
    },
    "KeyVaultEnableSoftDelete": {
      "type": "bool",
      "defaultValue": true,
      "metadata": {
        "description": "To be able to recover deleted objects"
      }
    },
    "KeyVaultCreateMode": {
      "type": "string",
      "allowedValues": [
        "default",
        "recover"
      ],
      "defaultValue": "default",
      "metadata": {
        "description": "To indicate whether the vault need to be recovered or not"
      }
    },
    "KeyVaultAccessPolicies": {
      "type": "array",
      "metadata": {
        "description": "Access Policy List"
      },
      "defaultValue": []
    },
    "KeyVaultEnableFirewall": {
      "type": "bool",
      "defaultValue": false,
      "metadata": {
        "description": "Indicates if the Firewall must be enabled for the Key Vault"
      }
    },
    "KeyVaultMicrosoftServicesBypassFirewall": {
      "type": "bool",
      "defaultValue": false,
      "metadata": {
        "description": "Indicates if must allow trusted Microsoft services to bypass the firewall"
      }
    },
    "KeyVaultFirewallAllowedSubnets": {
      "type": "array",
      "defaultValue": [],
      "metadata": {
        "description": "Information of existing virtual networks and subnets allowed to access the KeyVault"
      }
    },
    "KeyVaultFirewallAllowedPublicIPs": {
      "type": "array",
      "defaultValue": [],
      "metadata": {
        "description": "List of IPv4 IPs or ranges to allow access from the internet to the KeyVault"
      }
    },
```

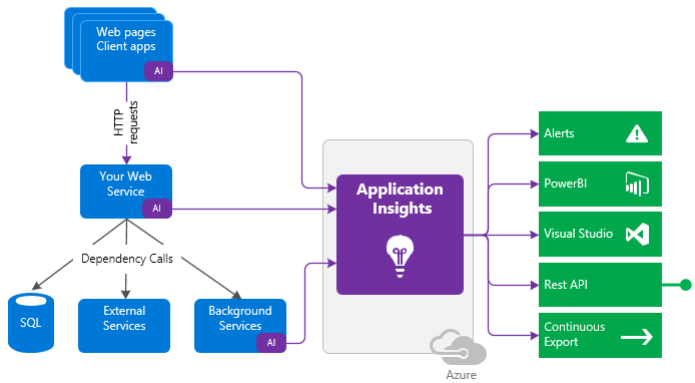
```
    "KeyVaultFirewallAllowedFirewallIps": {
      "type": "string",
      "defaultValue": "",
      "metadata": {
        "description": "Allowed firewall IPs",
        "AllEnvironmentsAutoAssignValueFromVariablesKeyVaultSecretName": "allowedFirewallIPs"
      }
    }
  },
  "variables": {
    "KeyVaultTenantId": "[subscription().tenantId]",
    "KeyVaultByPassFirewallTrue": "AzureServices",
    "KeyVaultByPassFirewallFalse": "None",
    "KeyVaultFirewallAllowedFirewallIpsArray": "[if(empty(parameters('KeyVaultFirewallAllowedFirewallIps')),variables('EmptyArray'),if(contains(parameters('KeyVaultFirewallAllowedFirewallIps'),','),split(parameters('KeyVaultFirewallAllowedFirewallIps'),'
    "KeyVaultFirewallAllIP": "[concat(if(empty(parameters('KeyVaultFirewallAllowedPublicIPs')),variables('EmptyArray'),parameters('KeyVaultFirewallAllowedPublicIPs')), variables('KeyVaultFirewallAllowedFirewallIpsArray'))]",
    "KeyVaultFirewallBypassAzureServices": "[if(parameters('KeyVaultMicrosoftServicesBypassFirewall'),variables('KeyVaultByPassFirewallTrue'),variables('KeyVaultByPassFirewallFalse'))]",
    "KeyVaultFirewallAllowedSubnetsArrayCount": "[length(parameters('KeyVaultFirewallAllowedSubnets'))]",
    "KeyVaultFirewallAllowedSubnetsArrayEmpty": "[equals(variables('KeyVaultFirewallAllowedSubnetsArrayCount'),0)]",
    "KeyVaultFirewallAllowedIPsArrayCount": "[length(variables('KeyVaultFirewallAllIP'))]",
    "KeyVaultFirewallAllowedIPsArrayEmpty": "[equals(variables('KeyVaultFirewallAllowedIPsArrayCount'),0)]",
    "copy": [
      {
        "name": "KeyVaultFirewallAllowedSubnets",
        "count": "[if(variables('KeyVaultFirewallAllowedSubnetsArrayEmpty'),1,variables('KeyVaultFirewallAllowedSubnetsArrayCount'))]",
        "input": {
          "id": "[if(variables('KeyVaultFirewallAllowedSubnetsArrayEmpty'),' ',concat('/subscriptions/',parameters('KeyVaultFirewallAllowedSubnets')[copyIndex('KeyVaultFirewallAllowedSubnets')].VNetSubscriptionId,'/resourceGroups/',parameters('KeyV
          "action": "Allow"
        }
      },
      {
        "name": "KeyVaultFirewallAllowedIpAddresses",
        "count": "[if(variables('KeyVaultFirewallAllowedIPsArrayEmpty'),1,variables('KeyVaultFirewallAllowedIPsArrayCount'))]",
        "input": {
          "value": "[if(variables('KeyVaultFirewallAllowedIPsArrayEmpty'),' ',variables('KeyVaultFirewallAllIP')[copyIndex('KeyVaultFirewallAllowedIpAddresses')])]",
          "action": "Allow"
        }
      }
    ],
    "EmptyArray": [],
    "firewallEnabled": {
      "bypass": "[variables('KeyVaultFirewallBypassAzureServices')]",
      "virtualNetworkRules": "[if(variables('KeyVaultFirewallAllowedSubnetsArrayEmpty'),variables('EmptyArray'),variables('KeyVaultFirewallAllowedSubnets'))]",
      "ipRules": "[if(variables('KeyVaultFirewallAllowedIPsArrayEmpty'),variables('EmptyArray'),variables('KeyVaultFirewallAllowedIpAddresses'))]",
      "defaultAction": "Deny"
    },
    "firewallDisabled": {
      "defaultAction": "Allow"
    }
  },
  "resources": [
    {
      "type": "Microsoft.KeyVault/vaults",
      "name": "[parameters('KeyVaultName')]",
      "apiVersion": "2016-10-01",
      "location": "[parameters('KeyVaultLocation')]",
      "properties": {
        "tenantId": "[variables('KeyVaultTenantId')]",
        "sku": {
          "family": "A",
          "name": "[parameters('KeyVaultSKUName')]"
        },
        "accessPolicies": "[parameters('KeyVaultAccessPolicies')]",
        "enabledForDeployment": "[parameters('KeyVaultEnabledForDeployment')]",
        "enabledForDiskEncryption": "[parameters('KeyVaultEnabledForDiskEncryption')]",
        "enabledForTemplateDeployment": "[parameters('KeyVaultEnabledForTemplateDeployment')]",
        "enableSoftDelete": "[if(parameters('KeyVaultEnableSoftDelete'), json('true'), json('null'))]",
        "createmode": "[parameters('KeyVaultCreatemode')]",
        "networkAcls": "[if(parameters('KeyVaultEnableFirewall'),variables('firewallEnabled'),variables('firewallDisabled'))]"
      }
    }
  ],
  "outputs": {}
}
```


Application Insights

Note

See Application Insights module in IaCs library

Application Insights, is an extensible Application Performance Management (APM) service for developers and DevOps professionals. Use it to monitor your live applications. It will automatically detect performance anomalies, and includes powerful analytics tools to help you diagnose issues and to understand what users actually do with your app. It's designed to help you continuously improve performance and usability. It works for apps on a wide variety of platforms including .NET, Node.js, Java, and Python hosted on-premises, hybrid, or any public cloud. It integrates with your DevOps process, and has connection points to a variety of development tools. It can monitor and analyze telemetry from mobile apps by integrating with Visual Studio App Center.



We use Application Insights to monitor the usage of our published documentation.

The ARM Template that we use for Application Insights deployment can be seen below. Notice that we output the value of the Application Insights, which is generated by Azure during the service deployment and cannot be anticipated, in two ways:

- As secret of the Key Vault passed as parameter
- As output of the ARM Template

This approach allows to reuse this value for configuration in future module deployments.

Note

See the Application Insights module that we use in our Evergreen Repository

```
{
  "$schema": "https://schema.management.azure.com/schemas/2019-04-01/deploymentTemplate.json#",
  "contentVersion": "1.0.0.0",
  "parameters": {
    "AppInsightsName": {
      "metadata": {
        "description": "The name of the Application Insights"
      },
      "type": "string"
    },
    "AppInsightsLocation": {
      "type": "string",
      "metadata": {
        "description": "The location for the Application Insights."
      }
    },
    "AppInsightsApplicationType": {
      "metadata": {
        "description": "The type of Application Insights"
      },
      "type": "string",
      "defaultValue": "web",
      "allowedValues": [
        "MobileCenter",
        "web",
        "other",
        "java",
        "Node.JS"
      ]
    },
    "AppInsightsKeyVaultName": {
      "type": "string",
      "metadata": {
        "description": "Name of the Key Vault where the Account's Connection Strings and Keys will be saved"
      },
      "defaultValue": ""
    },
    "AppInsightsSecretNameInstrumentationKey": {
      "type": "string",
      "defaultValue": "",
      "metadata": {
        "description": "Output secret name for primary master key"
      }
    }
  },
  "variables": {
    "AppInsightsSecretNameSecondaryMasterKey": "[if(not(empty(parameters('AppInsightsSecretNameInstrumentationKey'))), parameters('AppInsightsSecretNameInstrumentationKey'), concat(parameters('AppInsightsName'),'-instrumentation-key'))]"
  },
  "resources": [
    {
      "type": "Microsoft.Insights/components",
      "apiVersion": "2015-05-01",
      "name": "[parameters('AppInsightsName')]",
      "location": "[parameters('AppInsightsLocation')]",
      "kind": "[parameters('AppInsightsApplicationType')]",
      "properties": {
        "Application_Type": "[parameters('AppInsightsApplicationType')]",
        "Flow_Type": "Bluefield",
        "Request_Source": "rest"
      }
    },
    {
      "condition": "[not(empty(parameters('AppInsightskeyVaultName')))]",
      "type": "Microsoft.KeyVault/vaults/secrets",
      "name": "[concat(parameters('AppInsightskeyVaultName'), '/', variables('AppInsightsSecretNameSecondaryMasterKey'))]",
      "apiVersion": "2018-02-14",
      "properties": {
        "value": "[reference(parameters('AppInsightsName')).InstrumentationKey]",
        "contentType": "string"
      },
      "dependsOn": [
        "[concat('Microsoft.Insights/components/',parameters('AppInsightsName'))]"
      ]
    }
  ],
  "outputs": {
    "instrumentationKey": {
      "type": "string",
      "value": "[reference(parameters('AppInsightsName')).InstrumentationKey]"
    }
  }
}
```


App Service Plan

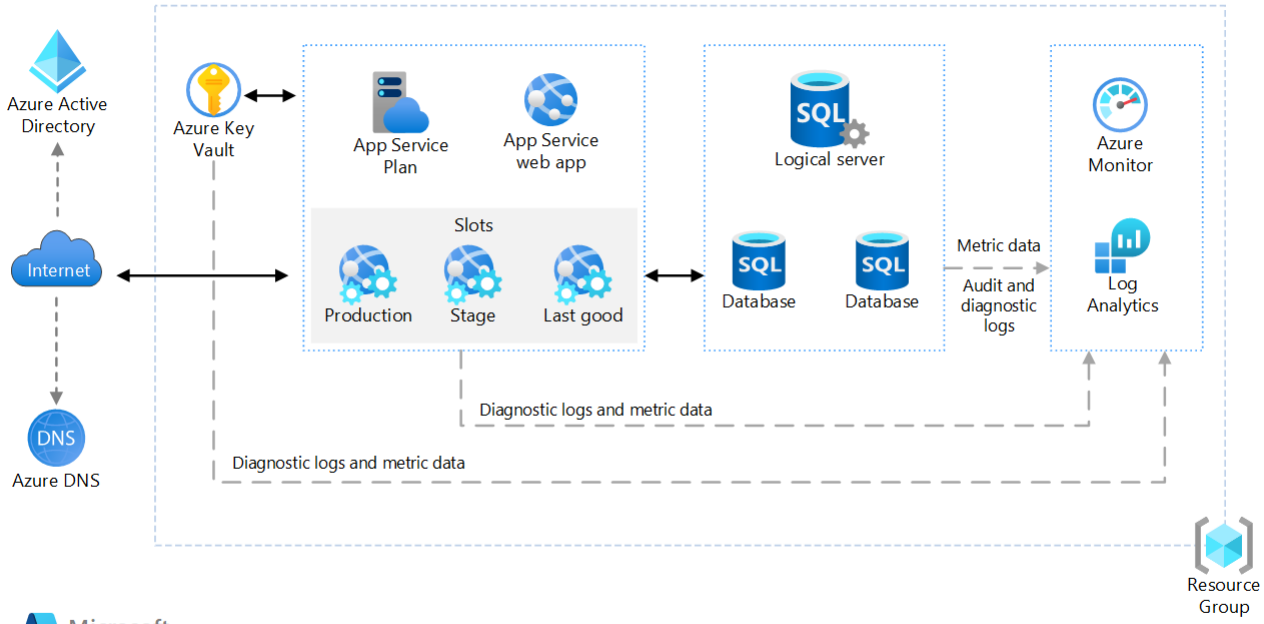
Note

See App Service Plan module in IaCs library

In App Service (Web Apps, API Apps, or Mobile Apps), an app always runs in an App Service plan. In addition, Azure Functions also has the option of running in an App Service plan. An App Service plan defines a set of compute resources for a web app to run. These compute resources are analogous to the server farm in conventional web hosting. One or more apps can be configured to run on the same computing resources (or in the same App Service plan).

When you create an[App Service plan in a certain region (for example, West Europe), a set of compute resources is created for that plan in that region. Whatever apps you put into this App Service plan run on these compute resources as defined by your App Service plan. Each App Service plan defines:

- Region (West US, East US, etc.)
- Number of VM instances
- Size of VM instances (Small, Medium, Large)
- Pricing tier (Free, Shared, Basic, Standard, Premium, PremiumV2, PremiumV3, Isolated)



We use App Service Plan to host the web sites of all our different environments

Note

See the App Service Plan module that we use in our Evergreen Repository

```
{
  "$schema": "https://schema.management.azure.com/schemas/2019-04-01/deploymentTemplate.json#",
  "contentVersion": "1.0.0.0",
  "parameters": {
    "WindowsAppServicePlanName": {
      "metadata": {
        "description": "The name of the App Service Plan that hosts the App Service"
      },
      "type": "string"
    },
    "WindowsAppServicePlanLocation": {
      "metadata": {
        "description": "The location where the App Service and the App Service Plan will be deployed"
      },
      "type": "string"
    },
    "WindowsAppServicePlanInstances": {
      "metadata": {
        "description": "The amount of instances that will run the App Services in the App Service Plan"
      },
      "type": "int"
    },
    "WindowsAppServicePlanTier": {
      "type": "string",
      "allowedValues": [
        "Free",
        "Shared",
        "Basic",
        "Standard",
        "Premium",
        "PremiumV2"
      ],
      "metadata": {
        "description": "The Tier of the App Service Plan. 'Free' and 'Shared' are shared plans that run Apps in shared Virtual Machines, with Apps from other customers. 'Basic', 'Standard', 'Premium' and 'PremiumV2' run apps on dedicated VMs"
      }
    },
    "WindowsAppServicePlanSize": {
      "type": "string",
      "allowedValues": [
        "1",
        "2",
        "3"
      ],
      "metadata": {
        "description": " The Size of the VMs that run the Apps. The higher this number is, the better the hardware of the underlying VMs is"
      }
    }
  },
  "variables": {
    "appServicePlanSizes": {
      "Free": {
        "SizeName": {
          "1": "F1",
          "2": "F1",
          "3": "F1"
        },
        "Family": "F"
      },
      "Shared": {
        "SizeName": {
          "1": "D1",
          "2": "D1",
          "3": "D1"
        },
        "Family": "D"
      },
      "Basic": {
        "SizeName": {
          "1": "B1",
          "2": "B2",
          "3": "B3"
        },
        "Family": "B"
      },
      "Standard": {
```

```
            "SizeName": {
                "1": "S1",
                "2": "S2",
                "3": "S3"
            },
            "Family": "S"
        },
        "Premium": {
            "SizeName": {
                "1": "P1",
                "2": "P2",
                "3": "P3"
            },
            "Family": "P"
        },
        "PremiumV2": {
            "SizeName": {
                "1": "P1v2",
                "2": "P2v2",
                "3": "P3v2"
            },
            "Family": "Pv2"
        }
    },
    "appServicePlanSkuSize": "[variables('appServicePlanSizes')[parameters('WindowsAppServicePlanTier')].SizeName[parameters('WindowsAppServicePlanSize')]]",
    "appServicePlanSkuFamily": "[variables('appServicePlanSizes')[parameters('WindowsAppServicePlanTier')].Family]",
    "appServicePlanSkuName": "[variables('appServicePlanSkuSize')]"
  },
  "resources": [
    {
      "type": "Microsoft.Web/serverfarms",
      "kind": "app",
      "name": "[parameters('WindowsAppServicePlanName')]",
      "apiVersion": "2016-09-01",
      "location": "[parameters('WindowsAppServicePlanLocation')]",
      "sku": {
        "name": "[variables('appServicePlanSkuName')]",
        "capacity": "[parameters('WindowsAppServicePlanInstances')]",
        "tier": "[parameters('WindowsAppServicePlanTier')]",
        "size": "[variables('appServicePlanSkuSize')]",
        "family": "[variables('appServicePlanSkuFamily')]"
      },
      "scale": null,
      "properties": {
        "name": "[parameters('WindowsAppServicePlanName')]",
        "perSiteScaling": false,
        "reserved": false,
        "targetWorkerCount": 0,
        "targetWorkerSizeId": 0
      },
      "dependsOn": [
      ]
    }
  ],
  "outputs": {
  }
}
```

App Service

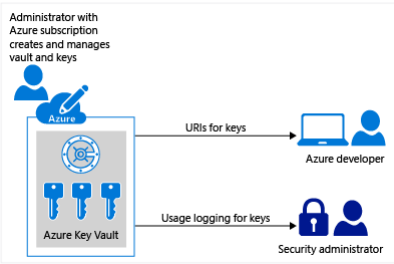
Note

See App Service module in IaCs library

Azure App Service is an HTTP-based service for hosting web applications, REST APIs, and mobile back ends. You can develop in your favorite language, be it .NET, .NET Core, Java, Ruby, Node.js, PHP, or Python. Applications run and scale with ease on both Windows and Linux-based environments.

App Service not only adds the power of Microsoft Azure to your application, such as security, load balancing, autoscaling, and automated management. You can also take advantage of its DevOps capabilities, such as continuous deployment from Azure DevOps, GitHub, Docker Hub, and other sources, package management, staging environments, custom domain, and TLS/SSL certificates.

With App Service, you pay for the Azure compute resources you use. The compute resources you use are determined by the App Service plan that you run your apps on.



We use App Service as the hosting web server for our website for each environment.

Note

See the App Service module that we use in our Evergreen Repository

```
{
  "$schema": "https://schema.management.azure.com/schemas/2019-04-01/deploymentTemplate.json#",
  "contentVersion": "1.0.0.0",
  "parameters": {
    "WindowsAppServicePlanName": {
      "metadata": {
        "description": "The name of the App Service Plan"
      },
      "type": "string"
    },
    "WindowsAppServiceName": {
      "metadata": {
        "description": "The name of the App Service"
      },
      "type": "string"
    },
    "WindowsAppServiceLocation": {
      "metadata": {
        "description": "The location of the App Service"
      },
      "type": "string"
    },
    "WindowsAppServiceAppInsightsInstrumentationKey": {
      "metadata": {
        "description": "The Application Insights Instrumentation Key of instance connected to the App Service"
      },
      "type": "string"
    },
    "WindowsAppServicePhpVersion": {
      "metadata": {
        "description": "The version of PHP on Windows Machines"
```

```
    },
    "type": "string",
    "defaultValue": ""
  },
  "WindowsAppServicePythonVersion": {
    "metadata": {
      "description": "The version of Python on Windows Machines"
    },
    "type": "string",
    "defaultValue": ""
  },
  "WindowsAppServiceNodeVersion": {
    "metadata": {
      "description": "The Version of Node.js on Windows Machines"
    },
    "type": "string",
    "defaultValue": "6.9"
  },
  "WindowsAppServiceJavaVersion": {
    "metadata": {
      "description": "The Java Version on Windows Machines"
    },
    "type": "string",
    "defaultValue": ""
  },
  "WindowsAppServiceJavaContainer": {
    "metadata": {
      "description": "The Java Container on Windows Machines"
    },
    "type": "string",
    "defaultValue": ""
  },
  "WindowsAppServiceJavaContainerVersion": {
    "metadata": {
      "description": "The Java Container Version on Windows Machines"
    },
    "type": "string",
    "defaultValue": ""
  },
  "WindowsAppServiceHTTPS": {
    "metadata": {
      "description": "Configures a web site to accept only https requests. Issues redirect for http requests",
      "mandatory": "true"
    },
    "type": "bool",
    "defaultValue": true
  },
  "WindowsAppServiceAlwaysOn": {
    "metadata": {
      "description": "Indicates that your web app needs to be loaded at all times. By default, web apps are unloaded after they have been idle. It is recommended that you enable this option when you have continuous web jobs running on the web ap
      "mandatory": "true"
    },
    "type": "bool",
    "defaultValue": true
  },
  "WindowsAppServiceAppSettings": {
    "metadata": {
      "description": "Define App Settings to be included.",
      "mandatory": "false"
    },
    "type": "object",
    "defaultValue": {
    }
  },
  "WindowsAppServiceAuthenticationADApplicationId": {
    "metadata": {
      "description": "These settings allow users to sign in with Azure Active Directory",
      "mandatory": "false"
    },
    "type": "string",
    "defaultValue": ""
  },
  "WindowsAppServiceManagedServiceIdentityEnabled": {
    "metadata": {
      "description": "Enables the usage of Managed Service Identities in App Service",
      "mandatory": "false"
    },
    "type": "bool",
    "defaultValue": true
  }
},
"variables": {
  "appServiceKind": "app",
  "AADApplicationSecret": "null",
  "azureADAuthenticationProperties": {
    "enabled": true,
    "httpApiPrefixPath": null,
    "unauthenticatedClientAction": 0,
    "tokenStoreEnabled": true,
    "allowedExternalRedirectUrls": null,
    "defaultProvider": 0,
    "clientId": "[parameters('WindowsAppServiceAuthenticationADApplicationId')]",
    "clientSecret": "[if(equals(variables('AADApplicationSecret'),'null'),json('null'),if(empty(variables('AADApplicationSecret')), json('null'), variables('AADApplicationSecret')))]",
    "issuer": "[concat('https://sts.windows.net/', subscription().tenantId)]",
    "allowedAudiences": null,
    "additionalLoginParams": null,
    "isAadAutoProvisioned": false
  },
  "nullAuthenticationProperties": {
    "enabled": false,
    "unauthenticatedClientAction": 1,
    "issuer": null,
    "clientId": null,
    "clientSecret": null
  },
  "nullValue": {
  },
  "appServiceOrServicePlan": "[if(bool(parameters('WindowsAppServiceManagedServiceIdentityEnabled')), parameters('WindowsAppServiceName'), parameters('WindowsAppServicePlanName'))]",
  "appServiceIdentityOrServicePlan": "[if(bool(parameters('WindowsAppServiceManagedServiceIdentityEnabled')), 'identity', 'name')]",
  "EmptyArray": [],
  "defaultAppSettings": {
    "APPINSIGHTS_INSTRUMENTATIONKEY": "[parameters('WindowsAppServiceAppInsightsInstrumentationKey')]",
    "WEBSITE_NODE_DEFAULT_VERSION": "[parameters('WindowsAppServiceNodeVersion')]"
  }
},
"resources": [
  {
    "type": "Microsoft.Web/sites",
    "kind": "[variables('appServiceKind')]",
    "name": "[concat(parameters('WindowsAppServiceName'))]",
    "apiVersion": "2018-02-01",
    "location": "[parameters('WindowsAppServiceLocation')]",
    "scale": null,
    "identity": {
      "type": "[if(parameters('WindowsAppServiceManagedServiceIdentityEnabled'), 'SystemAssigned', 'None')]"
    },
    "properties": {
      "name": "[parameters('WindowsAppServiceName')]",
      "serverFarmId": "[resourceId('Microsoft.Web/serverfarms', parameters('WindowsAppServicePlanName'))]",
      "reserved": false,
```

```
        "httpsOnly": "[parameters('WindowsAppServiceHTTPS')]"
      },
      "resources": [
        {
          "apiVersion": "2016-08-01",
          "name": "authsettings",
          "type": "config",
          "dependsOn": [
            "[concat('Microsoft.Web/sites/', parameters('WindowsAppServiceName'))]",
            "Microsoft.ApplicationInsights.AzureWebSites"
          ],
          "tags": {
            "displayName": "websiteAuthSettings"
          },
          "properties": "[if(empty(parameters('WindowsAppServiceAuthenticationADApplicationId')), variables('nullAuthenticationProperties'), variables('azureADAuthenticationProperties'))]"
        },
        {
          "name": "web",
          "type": "config",
          "apiVersion": "2020-09-01",
          "dependsOn": [
            "[concat('Microsoft.Web/sites/', parameters('WindowsAppServiceName'))]",
            "Microsoft.ApplicationInsights.AzureWebSites"
          ],
          "properties": {
            "phpVersion": "[parameters('WindowsAppServicePhpVersion')]",
            "pythonVersion": "[parameters('WindowsAppServicePythonVersion')]",
            "javaVersion": "[if(equals(parameters('WindowsAppServiceJavaVersion'),'Off'),'',parameters('WindowsAppServiceJavaVersion'))]",
            "javaContainer": "[if(equals(parameters('WindowsAppServiceJavaVersion'),'Off'),'',if(equals(parameters('WindowsAppServiceJavaVersion'),''),'',parameters('WindowsAppServiceJavaContainer')))]",
            "javaContainerVersion": "[if(equals(parameters('WindowsAppServiceJavaVersion'),'Off'),'',if(equals(parameters('WindowsAppServiceJavaVersion'),''),'',parameters('WindowsAppServiceJavaContainerVersion')))]",
            "minTlsVersion": "1.2",
            "ftpsState": "Disabled",
            "alwaysOn": "[parameters('WindowsAppServiceAlwaysOn')]",
            "httpLoggingEnabled": true,
            "requestTracingEnabled": true,
            "detailedErrorLoggingEnabled": true
          }
        },
        {
          "apiVersion": "2020-06-01",
          "name": "Microsoft.ApplicationInsights.AzureWebSites",
          "type": "siteextensions",
          "dependsOn": [
            "[resourceId('Microsoft.Web/sites', parameters('WindowsAppServiceName'))]"
          ],
          "properties": {
          }
        },
        {
          "type": "config",
          "apiVersion": "2020-09-01",
          "name": "appsettings",
          "dependsOn": [
            "[resourceId('Microsoft.Web/sites/', parameters('WindowsAppServiceName'))]",
            "Microsoft.ApplicationInsights.AzureWebSites"
          ],
          "properties": "[union(variables('defaultAppSettings'),parameters('WindowsAppServiceAppSettings'))]"

        }
      ]
    }
  ],
  "outputs": {
    "outboundIpAddresses": {
      "type": "string",
      "value": "[replace(replace(string(split(reference(parameters('WindowsAppServiceName')).outboundIpAddresses,',')),'[',''),']','')]"
    },
    "identity": {
      "type": "object",
      "value": "[if(parameters('WindowsAppServiceManagedServiceIdentityEnabled'), reference(variables('appServiceOrServicePlan'), '2018-02-01', 'Full')[variables('appServiceIdentityOrServicePlan')], variables('nullValue'))]"
    }
  }
}
```

Common environment configuration

The common environment will have the infrastructure services used commonly by all the other environments:

- Azure Key Vault to store configuration and secrets
- Application Insights to monitor the websites from all the environments
- App Service Plan to host websites from all the environments

We apply the following configuration for each module:

Azure Key Vault configuration

Note

See the Key Vault configuration that we use for the common environment in our Evergreen Repository

```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentParameters.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
      "KeyVaultName": {
        "value": "egipkv"
      },
      "KeyVaultLocation": {
        "value": "westeurope"
      },
      "KeyVaultSKUName": {
        "value": "standard"
      },
      "KeyVaultEnabledForDeployment": {
        "value": false
      },
      "KeyVaultEnabledForDiskEncryption": {
        "value": false
      },
      "KeyVaultEnabledForTemplateDeployment": {
        "value": true
      },
      "KeyVaultEnableSoftDelete": {
        "value": true
      },
      "KeyVaultCreateMode": {
        "value": "default"
      },
      "KeyVaultAccessPolicies": {
        "value": [
          {
            "tenantId": "72f988bf-86f1-41af-91ab-2d7cd011db47",
            "objectId": "8d839635-960b-49ae-8b72-ca2ac90c18ac",
            "permissions": {
              "keys": [ "all" ],
              "secrets": [ "all" ],
              "certificates": [ "all" ],
```

```
            "storage": []
          }
        }
      ]
    },
    "KeyVaultEnableFirewall": {
      "value": false
    }
  }
}
```

Application Insights configuration

Note

See the Application Insights configuration that we use for the common environment in our Evergreen Repository

```
{
  "$schema": "https://schema.management.azure.com/schemas/2019-04-01/deploymentParameters.json#",
  "contentVersion": "1.0.0.1",
  "parameters": {
    "AppInsightsName": {
      "value": "egipai"
    },
    "AppInsightsLocation": {
      "value": "WestEurope"
    },
    "AppInsightsApplicationType": {
      "value": "web"
    },
    "AppInsightsKeyVaultName": {
      "value": "egipkv"
    },
    "AppInsightsSecretNameInstrumentationKey": {
      "value": "egipai-instrumentation-key"
    }
  }
}
```

App Service Plan configuration

Note

See the App Service Plan configuration that we use for the common environment in our Evergreen Repository

```
{
  "$schema": "https://schema.management.azure.com/schemas/2019-04-01/deploymentParameters.json#",
  "contentVersion": "1.0.0.1",
  "parameters": {
    "WindowsAppServicePlanName": {
      "value": "egipsp"
    },
    "WindowsAppServicePlanLocation": {
      "value": "WestEurope"
    },
    "WindowsAppServicePlanInstances": {
      "value": 1
    },
    "WindowsAppServicePlanTier": {
      "value": "Basic"
    },
    "WindowsAppServicePlanSize": {
      "value": "1"
    }
  }
}
```

Integration environment configuration

App Service for integration configuration

Note

See the App Service configuration that we use for the integration environment in our Evergreen Repository

```
{
  "$schema": "https://schema.management.azure.com/schemas/2019-04-01/deploymentParameters.json#",
  "contentVersion": "1.0.0.1",
  "parameters": {
    "WindowsAppServiceName": {
      "value": "evergreenip-int"
    },
    "WindowsAppServicePlanName": {
      "value": "egipsp"
    },
    "WindowsAppServiceLocation": {
      "value": "WestEurope"
    },
    "WindowsAppServiceAppInsightsInstrumentationKey": {
      "reference": {
        "keyVault": {
          "id": "/subscriptions/4d186e10-e0a0-4555-a97f-7c8c2272d741/resourceGroups/Evergreen-Delivery-IP/providers/Microsoft.KeyVault/vaults/egipkv"
        },
        "secretName": "egipai-instrumentation-key"
      }
    },
    "WindowsAppServiceHTTPS": {
      "value": true
    },
    "WindowsAppServiceAlwaysOn": {
      "value": false
    },
    "WindowsAppServiceAuthenticationADApplicationId": {
      "value": "2309c8a4-4bce-42cf-923a-2d92ea5ccdd3"
    },
    "WindowsAppServiceManagedServiceIdentityEnabled": {
      "value": false
    }
  }
}
```

Staging environment configuration

App Service for staging configuration

Note

See the App Service configuration that we use for the staging environment in our Evergreen Repository

```
{
  "$schema": "https://schema.management.azure.com/schemas/2019-04-01/deploymentParameters.json#",
  "contentVersion": "1.0.0.1",
```

```
    "parameters": {
        "WindowsAppServiceName": {
            "value": "evergreenip-staging"
        },
        "WindowsAppServicePlanName": {
            "value": "egipsp"
        },
        "WindowsAppServiceLocation": {
            "value": "WestEurope"
        },
        "WindowsAppServiceAppInsightsInstrumentationKey": {
            "reference": {
                "keyVault": {
                    "id": "/subscriptions/4d186e10-e0a0-4555-a97f-7c8c2272d741/resourceGroups/Evergreen-Delivery-IP/providers/Microsoft.KeyVault/vaults/egipkv"
                },
                "secretName": "egipai-instrumentation-key"
            }
        },
        "WindowsAppServiceHTTPS": {
            "value": true
        },
        "WindowsAppServiceAlwaysOn": {
            "value": false
        },
        "WindowsAppServiceAuthenticationADApplicationId": {
            "value": "2309c8a4-4bce-42cf-923a-2d92ea5ccdd3"
        },
        "WindowsAppServiceManagedServiceIdentityEnabled": {
            "value": false
        }
    }
}
```


Production environment configuration

App Service for production configuration

Note

See the App Service configuration that we use for the production environment in our Evergreen Repository


```
{
    "$schema": "https://schema.management.azure.com/schemas/2019-04-01/deploymentParameters.json#",
    "contentVersion": "1.0.0.1",
    "parameters": {
        "WindowsAppServiceName": {
            "value": "evergreenip"
        },
        "WindowsAppServicePlanName": {
            "value": "egipsp"
        },
        "WindowsAppServiceLocation": {
            "value": "WestEurope"
        },
        "WindowsAppServiceAppInsightsInstrumentationKey": {
            "reference": {
                "keyVault": {
                    "id": "/subscriptions/4d186e10-e0a0-4555-a97f-7c8c2272d741/resourceGroups/Evergreen-Delivery-IP/providers/Microsoft.KeyVault/vaults/egipkv"
                },
                "secretName": "egipai-instrumentation-key"
            }
        },
        "WindowsAppServiceHTTPS": {
            "value": true
        },
        "WindowsAppServiceAlwaysOn": {
            "value": false
        },
        "WindowsAppServiceAuthenticationADApplicationId": {
            "value": "2309c8a4-4bce-42cf-923a-2d92ea5ccdd3"
        },
        "WindowsAppServiceManagedServiceIdentityEnabled": {
            "value": false
        }
    }
}
```

Overview

This section provides guidance to set-up Azure DevOps YAML Pipelines to configure:

- Continuous Integration and Continouos Delivery of your hosting infrastructure.
- Continuous Integration and Continouos Delivery of your web site.

Pipelines repository structure

We place all YAML pipeline assets under the /pipelines directory of our repository. The /platform directory contains all configuration files for our Infrastructure as Code modules (see Infrastructure as Code).

```
|– / (root)
   |– pipelines/
   |    |– iacs/
   |    |– infra/
   |    |   |– deploy.yml
   |    |   |– variables.yml
   |    |– scripts/
   |    |– website/
   |        |– deploy.yml
   |        |– variables.yml
   |– platform/
```

Where:

- pipelines/iacs/ directory contains all Infrastructure as Code modules that we will use to deploy our infrastructure. See Infrastructure as Code.
- pipelines/infra/ directory contains the YAML pipeline and YAML variables file to implement continuous integration and continuous delivery of the infrastructure. See Infrastructure Pipeline
- pipelines/scripts/ directory contains scripts that support automation steps for our pipelines
- pipelines/website/ directory contains the YAML pipeline and YAML variables file to implement continuous integration and continuous delivery of the web site. See Web Site Pipeline

Branch policies

Note

See our branch policy configuration for the main branch

Branch policies allow to implement the agreed branching strategy in any repository and branch.

For our feature branching strategy, we will protect our main branch with branch policies to have at least one person from our team review the Pull Request (peer review).

Additionally, we configure the Website pipeline as a validation pipeline for every Pull Request to the main branch updating files under docs/ directory. We use this configuration to implement the environment strategy and let the Pull Request reviewer use the staging website to validate the proposed changes to the documentation.

Infrastructure YAML Pipeline

Note

See our Infrastructure Deployment pipeline: pipelines/infra/deploy.yml

Infra YAML Pre-requisites

Azure Active Directory Application for Authentication Configuration

Currently, MSIT AD policy does not allowing Azure AD Service Principals to create other Azure AD Service Principals or Applications, but users can. Given that we will use an Azure AD Application to enable built-in Azure AD Authentication for App Service, we will need to manually create from the portal the Azure AD Application that we want to use for this.

Note

Follow instructions in article Quickstart: Register an application with the Microsoft identity platform to create the AD Application to use for App Service authentication

The AD Application needs to have the following configuration:

Setting	Value	Description	Example
Who can use the application	Accounts in this organizational directory only	Select this option if you're building an application for use only by users (or guests) in your tenant.	
Redirect URIs	https://[your-production-website-name].azurewebsites.net/.auth/login/aad/callback	Redirect URIs are required for Azure AD to redirect the user to the correct URL once authenticated	https://evergreenip.azurewebsites.net/.auth/login/aad/callback
	https://[your-staging-website-name].azurewebsites.net/.auth/login/aad/callback		https://evergreenip-staging.azurewebsites.net/.auth/login/aad/callback
	https://[your-integration-website-name].azurewebsites.net/.auth/login/aad/callback		https://evergreenip-int.azurewebsites.net/.auth/login/aad/callback
Select the tokens you would like to be issued by the authorization endpoint	ID tokens (used for implicit and hybrid flows)	These tokens will allow to capture id token logs in Application Insights to build the Insights dashboards	

Example of redirect configuration:

Example of ID tokens and tenant configuration:

Infra YAML name

Note

Suggested pipeline name format for detailed tracking: $(Build.SourceBranchName).$(Date:yyyyMMdd).$(Rev:rr)

Pipeline name is configured as follows in the YAML pipeline:

```
name: $(Build.SourceBranchName).$(Date:yyyyMMdd).$(Rev:rr)
```

Infra YAML triggers

Triggers allow to run the pipeline automatically under certain conditions.

Pipeline triggers are set as follows in the YAML pipeline:

```
trigger:
  branches:
    include:
      - main
  paths:
    include:
    - platform/ #ci on 'platform/*' files change
```

- The main trigger under branches will run the pipeline only for commits to the main branch.
- The platform/ trigger under paths will run the pipeline on every change to any file under platform/ path.

Triggers above will allow to run the pipeline automatically on every commit to the master branch that affect files under the platform/ directory, so that only changes to the infrastructure configuration files will trigger the pipeline.

Infra YAML variables

Variables can be defined in one YAML file and then imported in another pipeline. We leverage this strategy to allow for portability and reusability of the deployment pipeline.

Note

See our pipeline variables file: pipelines/infra/variables.yml

The following variables will be loaded from variables.yml file during pipeline execution:

```
variables:
  # pipeline config
  azureRMServiceConnection: 'Evergreen-Delivery-IP' # The ADO Service Connection to be used for the deployment
  subscriptionId: '4d186e10-e0a0-4555-a97f-7c8c2272d741' #T he subscription ID where Infrastructure will be deployed
  vmImageName: 'ubuntu-latest'  # Agent VM image name
  System.Debug: true # View pipeline logs
  configFilesPath: '$(Build.SourcesDirectory)/platform' # Path to platform IaC configuration files
  iacsTemplatesPath: '$(Build.SourcesDirectory)/pipelines/iacs' # Path to IaC modules path
  keyVaultTemplateFilePath: '$(iacsTemplatesPath)/key-vault.json' # Path to Key Vault module deployment template file
  applicationInsightsTemplateFilePath: '$(iacsTemplatesPath)/application-insights.json' #  Path to Application Insights module deployment template file
  appServicePlanWindowsTemplateFilePath: '$(iacsTemplatesPath)/app-service-plan-windows.json' # Path to Windows App Service Plan module deployment template file
  appServiceWindowsTemplateFilePath: '$(iacsTemplatesPath)/app-service-windows.json' # Path to Windows App Service module deployment template file

  # environment config
  configKeyVaultName: 'egipkv' # Name of the Key Vault used to store configuration variables
  resourceGroupName: 'Evergreen-Delivery-IP' # Name of the Resource Group where the infrastructure will be deployed
  resourceGroupLocation: 'West Europe' # Location of the Resource Group where the infrastructure will be deployed

  # common
  commonEnvironmentName: '$(resourceGroupName)-common' # Name given to the Azure pipelines deployment environment with deployment steps for the common hosting infrastructure elements, listed below
  configFilePathKeyVault: '$(configFilesPath)/common/egipkv.json' # Path to the Key Vault configuration file used to store configuration variables
  configFilePathApplicationInsights: '$(configFilesPath)/common/egipai.json' # Path to the Application Insights configuration file used to capture web site logs
  configFilePathAppServicePlanWindows: '$(configFilesPath)/common/egipsp.json' # Path to the Windows App Service Plan configuration file used to host the App Services for all environments

  # int
  intEnvironmentName: 'infra-$(resourceGroupName)-int' # Name given to the Azure pipelines deployment environment with deployment steps for the integration hosting infrastructure elements, listed below
  intConfigFilePathAppServiceWindows: '$(configFilesPath)/int/evergreenip-int.json' # Path to the Windows App Service configuration file used to host the web app for the integration environment

  # staging
  stagingEnvironmentName: 'infra-$(resourceGroupName)-staging' # Name given to the Azure pipelines deployment environment with deployment steps for the staging hosting infrastructure elements, listed below
  stagingConfigFilePathAppServiceWindows: '$(configFilesPath)/staging/evergreenip-staging.json' # Path to the Windows App Service configuration file used to host the web app for the staging environment

  # prod
  prodEnvironmentName: 'infra-(resourceGroupName)-prod' # Name given to the Azure pipelines deployment environment with deployment steps for the production hosting infrastructure elements, listed below
  prodConfigFilePathAppServiceWindows: '$(configFilesPath)/prod/evergreenip.json' # Path to the Windows App Service configuration file used to host the web app for the production environment
```

Pipeline variables are configured as follows in the YAML pipeline:

```
variables:
  - template: variables.yml
  - name: isManual
    value: $[eq(variables['Build.Reason'], 'Manual')]
  - name: isPullRequest
    value: $[eq(variables['Build.Reason'], 'PullRequest')]
  - name: isMain
    value: $[eq(variables['Build.SourceBranch'], 'refs/heads/main')]
  - name: isMainPR
    value: $[and(eq(variables['isPullRequest'],'True'),eq(variables['System.PullRequest.TargetBranch'], 'refs/heads/main'))]
```

Additionally, the following variables calculated at run time are added:

- isManual: true if the event that caused the build to run is a manual user trigger, false otherwise.
- isPullRequest: true if the event that caused the build to run is a Git branch policy that requires a build to run, false otherwise
- isMain: true if the source branch is refs/head/main, false otherwise. Used as condition to run deployment steps only when there is a change in the main branch.
- isMainPR: true if the target branch is refs/head/main and the isPullRequest condition is met , false otherwise. Used as condition to run deployment steps only when there is a Pull Request to the main branch.

Infra YAML build steps

No build steps will be executed for Infrastructure deployment. Build steps are included in the IaC modules creation to validate the quality of the IaC modules. See How to create a Module? in IaCS.

Infra YAML deployment steps

The deployment steps will allow deploy the Infrastructure elements to Azure for all the environments. The strategy to follow for the deployment is:

- Each environment will be deployed in its own stage
- Each stage will use deployment jobs to orchestrate the deployment
- Certain configuration will be generated at deployment time and cannot be predicted beforehand (e.g. Application Insights Instrumentation Key) and needs to be reused in future deployment jobs. Two strategies can be followed:
  - Define task output variables and use those variables to replace the values of the configuration files (using tasks such as Replace Tokens).
  - Leverage Azure Key Vault in the ARM Template to store the required configuration as Key Vault secrets that can be reused by other ARM Templates via ARM Template secret references

Common infra stage

Steps to deploy the common infrastructure are configured as follows in the YAML pipeline:

```
stages:
  - stage: common
    displayName: "common"
    dependsOn: []
    jobs:
      - deployment: common
        displayName: "common infrastructure"
        pool:
```

```
      vmImage: $(vmImageName)
    environment: $(commonEnvironmentName)  # target environment name and optionally a resource name to record the deployment history; format: <environment-name>.<resource-name>
    strategy:
      runOnce: #rolling, canary are the other strategies that are supported
        deploy:
          steps:
            - checkout: self
            # deploy key vault to use secrets as arm template configuration
            - task: AzureResourceManagerTemplateDeployment@3
              displayName: 'key vault'
              inputs:
                deploymentScope: 'Resource Group'
                azureResourceManagerConnection: $(azureRMServiceConnection)
                subscriptionId: $(subscriptionId)
                action: 'Create Or Update Resource Group'
                resourceGroupName: $(resourceGroupName)
                location: $(resourceGroupLocation)
                templateLocation: 'Linked artifact'
                csmFile: '$(keyVaultTemplateFilePath)'
                csmParametersFile: '$(configFilePathKeyVault)'
                deploymentMode: 'Incremental'
                deploymentOutputs: 'OUTPUTS_KEYVAULT'
                pwsh: true
            # deploy application insights
            - task: AzureResourceManagerTemplateDeployment@3
              displayName: 'application insights'
              inputs:
                deploymentScope: 'Resource Group'
                azureResourceManagerConnection: $(azureRMServiceConnection)
                subscriptionId: $(subscriptionId)
                action: 'Create Or Update Resource Group'
                resourceGroupName: $(resourceGroupName)
                location: $(resourceGroupLocation)
                templateLocation: 'Linked artifact'
                csmFile: '$(applicationInsightsTemplateFilePath)'
                csmParametersFile: '$(configFilePathApplicationInsights)'
                deploymentMode: 'Incremental'
                deploymentOutputs: 'OUTPUTS_APPLICATIONINSIGHTS'
                pwsh: true
            # deploy windows app service plan
            - task: AzureResourceManagerTemplateDeployment@3
              displayName: 'windows asp'
              inputs:
                deploymentScope: "Resource Group"
                azureResourceManagerConnection: $(azureRMServiceConnection)
                subscriptionId: $(subscriptionId)
                action: "Create Or Update Resource Group"
                resourceGroupName: "$(resourceGroupName)"
                location: $(resourceGroupLocation)
                templateLocation: "Linked artifact"
                csmFile: "$(appServicePlanWindowsTemplateFilePath)"
                csmParametersFile: "$(configFilePathAppServicePlanWindows)"
                deploymentMode: "Incremental"
                deploymentOutputs: "OUTPUTS_APPSERVICEPLANWINDOWS"
```

These deployment steps will deploy the following Azure Infrastructure:

- Key Vault to be used as configuration store
- Application insights to be used to capture metrics and monitoring. The instrumentation key will be stored as a secret in the Key Vault above
- Windows App Service Plan to host all the App Services

Because no conditions have been added to this stage, it will be executed every time the pipeline is triggered (via triggers configuration or manually)

Integration infra stage

Steps to deploy the integration infrastructure are configured as follows in the YAML pipeline:

```
  - stage: integration
    dependsOn:
    - common
    displayName: "webapp - int"
    jobs:
      - deployment: int
        displayName: "deploy windows webapp"
        pool:
          vmImage: $(vmImageName)
        environment: $(intEnvironmentName)  # target environment name and optionally a resource name to record the deployment history; format: <environment-name>.<resource-name>
        strategy:
          runOnce:    #rolling, canary are the other strategies that are supported
            deploy:
              steps:
                - checkout: self
                # deploy linux web app
                - task: AzureResourceManagerTemplateDeployment@3
                  displayName: 'app service'
                  inputs:
                    deploymentScope: 'Resource Group'
                    azureResourceManagerConnection: $(azureRMServiceConnection)
                    subscriptionId: $(subscriptionId)
                    action: 'Create Or Update Resource Group'
                    resourceGroupName: $(resourceGroupName)
                    location: $(resourceGroupLocation)
                    templateLocation: 'Linked artifact'
                    csmFile: '$(appServiceWindowsTemplateFilePath)'
                    csmParametersFile: '$(intConfigFilePathAppServiceWindows)'
                    deploymentMode: 'Incremental'
                    deploymentOutputs: 'OUTPUTS_WEBAPP_INT'
```

These deployment steps will deploy the following Azure Infrastructure:

- Windows App Service for the Integration environment. The ARM Template is configured to retrieve the Application Insights Instrumentation Key from the Key Vault secret deployed to the common infrastructure

Because no conditions have been added to this stage, it will be executed every time the pipeline is triggered (via triggers configuration or manually)

Staging infra stage

Steps to deploy the staging infrastructure are configured as follows in the YAML pipeline:

```
  - stage: staging
    dependsOn:
    - common
    - integration
    displayName: "webapp - staging"
    jobs:
      - deployment: int
        displayName: "deploy windows webapp"
        pool:
          vmImage: $(vmImageName)
```

```
    environment: $(stagingEnvironmentName)  # target environment name and optionally a resource name to record the deployment history; format: <environment-name>.<resource-name>
    strategy:
      runOnce:    #rolling, canary are the other strategies that are supported
        deploy:
          steps:
            - checkout: self
            # deploy linux web app
            - task: AzureResourceManagerTemplateDeployment@3
              displayName: 'app service'
              inputs:
                deploymentScope: 'Resource Group'
                azureResourceManagerConnection: $(azureRMServiceConnection)
                subscriptionId: $(subscriptionId)
                action: 'Create Or Update Resource Group'
                resourceGroupName: $(resourceGroupName)
                location: $(resourceGroupLocation)
                templateLocation: 'Linked artifact'
                csmFile: '$(appServiceWindowsTemplateFilePath)'
                csmParametersFile: '$(stagingConfigFilePathAppServiceWindows)'
                deploymentMode: 'Incremental'
                deploymentOutputs: 'OUTPUTS_WEBAPP_STAGING'
```

These deployment steps will deploy the following Azure Infrastructure:

- Windows App Service for the Staging environment. The ARM Template is configured to retrieve the Application Insights Instrumentation Key from the Key Vault secret deployed to the common infrastructure

Because no conditions have been added to this stage, it will be executed every time the pipeline is triggered (via triggers configuration or manually)

Production infra stage

Steps to deploy the production infrastructure are configured as follows in the YAML pipeline:

```
  - stage: prod
    condition: and(succeeded(), eq(variables.isMain, true)) # deploy only for main branch, after peer PR check and approval
    dependsOn:
    - common
    - integration
    - staging
    displayName: "webapp - prod"
    jobs:
      - deployment: int
        displayName: "deploy windows webapp"
        pool:
          vmImage: $(vmImageName)
        environment: $(prodEnvironmentName)  # target environment name and optionally a resource name to record the deployment history; format: <environment-name>.<resource-name>
        strategy:
          runOnce:    #rolling, canary are the other strategies that are supported
            deploy:
              steps:
                - checkout: self
                # deploy linux web app
                - task: AzureResourceManagerTemplateDeployment@3
                  displayName: 'app service'
                  inputs:
                    deploymentScope: 'Resource Group'
                    azureResourceManagerConnection: $(azureRMServiceConnection)
                    subscriptionId: $(subscriptionId)
                    action: 'Create Or Update Resource Group'
                    resourceGroupName: $(resourceGroupName)
                    location: $(resourceGroupLocation)
                    templateLocation: 'Linked artifact'
                    csmFile: '$(appServiceWindowsTemplateFilePath)'
                    csmParametersFile: '$(prodConfigFilePathAppServiceWindows)'
                    deploymentMode: 'Incremental'
                    deploymentOutputs: 'OUTPUTS_WEBAPP_PROD'
```

These deployment steps will deploy the following Azure Infrastructure:

- Windows App Service for the Production environment. The ARM Template is configured to retrieve the Application Insights Instrumentation Key from the Key Vault secret deployed to the common infrastructure

Because no conditions have been added to this stage, it will be executed every time the pipeline is triggered (via triggers configuration or manually)

Website YAML Pipeline

Note

See our Website Deployment pipeline: pipelines/website/deploy.yml

Website YAML pre-requisites

No pre-requisites are required for the Website Pipeline

Website YAML name

Note

Suggested pipeline name format for detailed tracking: $(Build.SourceBranchName).$(Date:yyyyMMdd).$(Rev:rr)

Pipeline name is configured as follows in the YAML pipeline:

```
name: $(Build.SourceBranchName).$(Date:yyyyMMdd).$(Rev:rr)
```

Website YAML Triggers

Triggers allow to run the pipeline automatically under certain conditions.

Pipeline triggers are set as follows in the YAML pipeline:

```
trigger:
  branches:
    include:
      - "*"
  paths:
    include:
    - docs/ #ci on 'docs/*' files change
```

- The * trigger under branches will run the pipeline only for commits to any branch.
- The docs/ trigger under paths will run the pipeline on every change to any file under docs/ path.

Triggers above will allow to run the pipeline automatically on every commit to any branch that affect files under the docs/ directory, so that only in changes to the docs articles will trigger the

pipeline.

Website YAML variables

Variables can be defined in one YAML file and then imported in another pipeline. We leverage this strategy to allow for portability and reusability of the deployment pipeline.

Note

See our pipeline variables file: pipelines/website/variables.yml

The following variables will be loaded from variables.yml file during pipeline execution:

```
variables:
  # pipeline config
  azureRMServiceConnection: 'Evergreen-Delivery-IP' #The ADO Service Connection to use for the deployment
  buildVmImageName: 'windows-latest'  # Agent VM image name for build jobs
  releaseVmImageName: 'ubuntu-latest'  # Agent VM image name for release jobs
  System.Debug: true # View pipeline logs

  # DocFX config
  docFXVersion: '2.57.2' # Check: https://github.com/dotnet/docfx/releases
  docFXOutputFolder: '$(Build.SourcesDirectory)/website' # The path where docfx will generate the website
  docFXZipFileName: 'website.zip' # The name of the zip file with the website
  docFXArtifactName: 'egip-website' # The ADO artifact name containing the website to be deployed

  # Website deployment package

  # environment config
  resourceGroupName: 'Evergreen-Delivery-IP' # The name of the RG where the App Service is deployed

  # int
  intEnvironmentName: 'website-$(resourceGroupName)-int' # The name of the Pipeline environment with the deployment steps for the Int environment
  intWebsiteName: 'evergreenip-int' # The name of the App Service for the Int environment

  # staging
  stagingEnvironmentName: 'website-$(resourceGroupName)-staging' # The name of the Pipeline environment with the deployment steps for the Staging environment
  stagingWebsiteName: 'evergreenip-staging' # The name of the App Service for the Staging environment

  # prod
  prodEnvironmentName: 'website-$(resourceGroupName)-prod' # The name of the Pipeline environment with the deployment steps for the Production environment
  prodWebsiteName: 'evergreenip' # The name of the App Service for the Production environment
```

Pipeline variables are configured as follows in the YAML pipeline:

```
variables:
  - template: variables.yml
  - name: isManual
    value: $[eq(variables['Build.Reason'], 'Manual')]
  - name: isPullRequest
    value: $[eq(variables['Build.Reason'], 'PullRequest')]
  - name: isMain
    value: $[eq(variables['Build.SourceBranch'], 'refs/heads/main')]
  - name: isMainPR
    value: $[and(eq(variables['isPullRequest'],'True'),eq(variables['System.PullRequest.TargetBranch'], 'refs/heads/main'))]
```

Additionally, the following variables calculated at run time are added:

- isManual: true if the event that caused the build to run is a manual user trigger, false otherwise.
- isPullRequest: true if the event that caused the build to run is a Git branch policy that requires a build to run, false otherwise
- isMain: true if the source branch is refs/head/main, false otherwise. Used as condition to run deployment steps only when there is a change in the main branch.
- isMainPR: true if the target branch is refs/head/main and the isPullRequest condition is met , false otherwise. Used as condition to run deployment steps only when there is a Pull Request to the main branch.

Website YAML build steps

The build steps for the website will allow to build the website from source code using DocFX tool and do any kind of static code analysis (e.g. Grammar Check, Markdown Linter, etc.) or Unit Testing before deploying the website.)

```
stages:
  - stage: build
    displayName: "build"
    dependsOn: []
    jobs:
      - job: docFXBuild
        displayName: "docFX build"
        pool:
          vmImage: $(buildVmImageName)
        steps:
        # build docfx site
        - powershell: |
            choco install docfx -y --version $(docFXVersion)
            docfx docfx.json -o '$(docFXOutputFolder)'
            if ($lastexitcode -ne 0){
                throw ("Error generating document")
            }
          displayName: 'docfx build'
        # zip docfx site
        - task: ArchiveFiles@2
          displayName: 'zip website'
          inputs:
            rootFolderOrFile: $(docFXOutputFolder)
            includeRootFolder: false
            archiveFile: '$(Build.ArtifactStagingDirectory)/$(docFXZipFileName)'
        # upload docfx site as pipeline artifact
        - task: PublishPipelineArtifact@1
          displayName: 'publish artifact'
          inputs:
            targetPath: '$(Build.ArtifactStagingDirectory)/$(docFXZipFileName)' #Path to the folder or file you want to publish. The path must be a fully qualified path or a valid path relative to the root directory of your repository. (Required)
            artifactName: '$(docFXArtifactName)' #Your artifact name. You can specify any name you prefer
            artifactType: 'pipeline' # Required. Options: pipeline, filepath. Default value: pipeline
```

These build steps will achieve the following:

- Step docfx build uses the DocFX tool to generate the website from source code.
- Step zip website packages the generated website into a zip file.
- Step publish artifact publishes the zip file as a pipeline artifact that can be deployed in future jobs to multiple environments.

Because no conditions have been added to this stage, it will be executed every time the pipeline is triggered (via triggers configuration or manually)

Website YAML deployment steps

The deployment steps will deploy the web to the app services for all the environments. The strategy to follow for the deployment is:

- Each environment will be deployed in its own stage

- Each stage will use deployment jobs to orchestrate the deployment

Website integration stage

Steps to deploy the website to the integration stage are configured as follows in the YAML pipeline:

```
- stage: integration
  dependsOn:
  - build
  displayName: "webapp - int"
  jobs:
    - deployment: int
      displayName: "deploy windows webapp"
      pool:
        vmImage: $(releaseVmImageName)
      environment: $(intEnvironmentName)  # target environment name and optionally a resource name to record the deployment history; format: <environment-name>.<resource-name>
      strategy:
        runOnce:    #rolling, canary are the other strategies that are supported
          deploy:
            steps:
            # download docfx artifact
            - task: DownloadPipelineArtifact@2
              displayName: 'download docfx artifact'
              inputs:
                source: 'current' # Download artifacts produced by the current pipeline run, or from a specific pipeline run. Options: current, specific
                path: '$(Pipeline.Workspace)'
            # deploy webapp
            - task: AzureRmWebAppDeployment@4
              displayName: 'deploy webapp - int'
              inputs:
                azureSubscription: $(azureRMServiceConnection)
                WebAppName: $(intWebsiteName)
                Package: '$(Pipeline.Workspace)/$(docFXArtifactName)/$(docFXZipFileName)'
                enableCustomDeployment: true
                DeploymentType: zipDeploy
                TakeAppOfflineFlag: false
```

These deployment steps will do the following:

- The `download docfx artifact` step will download the artifact generated during the build phase
- The `deploy webapp - int` step will deploy the zip file containing the website to the app service in the integration environment

Because no conditions have been added to this stage, it will be executed every time the pipeline is triggered (via triggers configuration or manually), and after the `build` stage successfully finishes as declared in the `dependsOn` section.

Website staging stage

Steps to deploy the website to the staging stage are configured as follows in the YAML pipeline:

```
- stage: staging
  condition: and(succeeded(), eq(variables.isMainPR, true)) # deploy only for Pull Request to main branch
  dependsOn:
  - build
  - integration
  displayName: "webapp - staging"
  jobs:
    - deployment: staging
      displayName: "deploy windows webapp"
      pool:
        vmImage: $(releaseVmImageName)
      environment: $(stagingEnvironmentName)  # target environment name and optionally a resource name to record the deployment history; format: <environment-name>.<resource-name>
      strategy:
        runOnce:    #rolling, canary are the other strategies that are supported
          deploy:
            steps:
            # download docfx artifact
            - task: DownloadPipelineArtifact@2
              displayName: 'download docfx artifact'
              inputs:
                source: 'current' # Download artifacts produced by the current pipeline run, or from a specific pipeline run. Options: current, specific
                path: '$(Pipeline.Workspace)'
            # deploy webapp
            - task: AzureRmWebAppDeployment@4
              displayName: 'deploy webapp - staging'
              inputs:
                azureSubscription: $(azureRMServiceConnection)
                WebAppName: $(stagingWebsiteName)
                Package: '$(Pipeline.Workspace)/$(docFXArtifactName)/$(docFXZipFileName)'
                enableCustomDeployment: true
                DeploymentType: zipDeploy
                TakeAppOfflineFlag: false
```

These deployment steps will do the following:

- The `download docfx artifact` step will download the artifact generated during the build phase
- The `deploy webapp - staging` step will deploy the zip file containing the website to the app service in the staging environment

The condition `and(succeeded(), eq(variables.isMainPR, true))` will make this stage to run only for Pull Requests to the `main` branch and after the `build` stage successfully finishes as declared in the `dependsOn` section.

Website production stage

Steps to deploy the website to the production stage are configured as follows in the YAML pipeline:

```
- stage: prod
  condition: and(succeeded(), eq(variables.isMain, true)) # deploy only for main branch
  dependsOn:
  - build
  displayName: "webapp - prod"
  jobs:
    - deployment: prod
      displayName: "deploy windows webapp"
      pool:
        vmImage: $(releaseVmImageName)
      environment: $(prodEnvironmentName)  # target environment name and optionally a resource name to record the deployment history; format: <environment-name>.<resource-name>
      strategy:
        runOnce:    #rolling, canary are the other strategies that are supported
          deploy:
            steps:
            # download docfx artifact
            - task: DownloadPipelineArtifact@2
              displayName: 'download docfx artifact'
              inputs:
                source: 'current' # Download artifacts produced by the current pipeline run, or from a specific pipeline run. Options: current, specific
                path: '$(Pipeline.Workspace)'
            # deploy webapp
            - task: AzureRmWebAppDeployment@4
```

```
    displayName: 'deploy webapp - production'
    inputs:
      azureSubscription: $(azureRMServiceConnection)
      WebAppName: $(prodWebsiteName)
      Package: '$(Pipeline.Workspace)/$(docFXArtifactName)/$(docFXZipFileName)'
      enableCustomDeployment: true
      DeploymentType: zipDeploy
      TakeAppOfflineFlag: false
```

These deployment steps will do the following:

- The `download docfx artifact` step will download the artifact generated during the build phase
- The `deploy webapp - production` step will deploy the zip file containing the website to the app service in the production environment

The condition `and(succeeded(), eq(variables.isMain, true))` will make this stage to run only for commits to the `main` branch and after the `build` stage successfully finishes as declared in the `dependsOn` section.

Azure DevOps Pipeline configuration

Once the YAML pipeline is edited, and Azure DevOps Pipeline needs to be configured to run the YAML file as configured. This pipeline in Azure DevOps will allow to:

- See history of pipeline executions and results
- Trigger the pipeline manually or as define in the triggers configuration
- Visualize deployment status for the different environments and tasks

See article Multi-stage pipelines user experience.

Note

See our pipeline configuration in the ADO portal: evergreen-infra and evergreen-website

Example of YAML infrastructure pipeline configuration:

Example of pipeline history:

Example of pipeline execution status:

Overview

This section provides guidance to set-up Azure DevOps Graphical User Interface Pipelines to configure:

- Continuous Integration and Continouos Delivery of your site hosting infrastructure.
- Continuous Integration and Continouos Delivery of your site web application.

Infrastructure CI/CD

Approach

The infrastructure required for self-hosting the documentation web site and also having app insights based insights into the usage of the site contains:

- Azure App Service Plan used to provision environment for hosting web site
- Azure Web App to host the web site itself
- Azure Monitor AppInsights to track the usage of the site
- 2 app registrations:
  - one to get access to Azure to be able to deploy the solution into resource group (need contributor permissions assigned), configured Service Connections settings of the ADO
  - another one to host the web site (and register it with Azure AD of MSIT for single sign-on)

The approach for self-service site hosting provides configuration modules from the infrastructure as code work and therefore relies on build and release pipelines dependant on these base modules and configuration files.

Note

The file examples used for the deployment of this documentation site are available as a starter kit in Evergreen Repo => /platform/iac-modules and /platform/prod.

To see the entire infrastructure configured including provisioning application registrations, please watch the video below.

Infrastructure Build Pipeline

Using the configuration modules and the configuration files build and release pipelines can be configured for the deployment of initial version of the web app and later for deployment of changes. The build pipeline's main task it to get those module files and configuration scripts and publish them as artifact for release pipeline to be able to use.

Infrastructure Release Pipeline

Release pipeline is more complex as it contains multiple deployment steps:

1. Deploy app insights
2. Extract app insights ID (leveraging third party build pipeline task ARM outputs that needs to be installed in organization hosting the pipelines/repo)
3. Deploy app service plan to host the web site
4. Deploy the app itself

Note

Suggested release name format for detailed tracking: $(Release.DefinitionName)-$(Date:yyyyMMdd)$(Rev:.r)

Warning

Note also that because MSIT AD is locked down in a specific way and does not allow service account provisioning and linking from web apps automatically, there need to be two steps performed as part of initial deployment: (1) deployment of the web site without the WindowsAppServiceAuthenticationADApplicationId parameter being specified (this will mean the site is anonymous initially) and (2) after getting the application ID once more with this parameter containing the value provided by ID for the application.

Web Site CI/CD

Web Site Approach

Build and deployment of the documentation web site leverages previously provisioned infrastructure.

Web Site Build Pipeline

The build process consists of these steps:

1. Building the DOCFX web site
2. Packaging the web site built
3. Publishing the artifact to that release pipelines can pick it up and deploy

Warning

When packaging the documentation site, remove the prepend root folder name to achieve paths to ensure that your web site's root will be root of the application. We had first missed this in the video below.

Web Site Release Pipeline

Web site release pipeline is simpler because the only thing that it does is it uses standard web app deploy task to push the ZIP file created to the web site.

Example of Configuration

For a walk-through of these changes being made to existing repository and introducing both initial DOCFX configuration as well as setting up the infrastructure configuration and CI/CD for deploying the documentation site see the video below.

The video shows also some "troubleshooting" of AD integration configuration and one build pipeline setting as we uncover and remove some blockers (aka 'evergivens').

The example files for pipelines, TOC, DOCFX JSON used in the video as starters that you can modify for your pipelines you can find part of the evergreen delivery ip docs repository.

Insights Overview

What you cannot see, you cannot measure. What you cannot measure, you cannot improve. This classic management axiom is true in the innersource world as well. If we want to make our project a success, it is important that we understand whether what we are creating works, but more importantly, how it works.

In the same way that we **monitor** the activity of an ordinary web page, we must also monitor the behaviour in an innersource project. Furthermore, this is one of the pillars of innersourcing.

In this section we will explain **why** it is important to monitor **what** happens behind the scenes and upfront, what aspects should we be taken care of and **how** we can do it.

Why

Monitoring provides us with tools and mechanisms that help us **measure the impact** of the content we are producing. Understanding when there has been a service failure, when a user abandons a page or what is the flow that a user follows on our website, allows us to establish patterns that will help us to continuously improve the project.

On the other hand, it is important to understand for what purpose the content is being consumed. If we are talking about an offering, for example, knowing who is visiting our website and what projects they are working on is critical when it comes to identifying potential opportunities with customers.

There are two main perspectives from which to monitor an innersource project and both are equally important.

- **Consumption perspective**: at the end of the day, the goal of the innersource project is for people to **consume and reuse** the content. That's why, just as you monitor users and their behavior on a website, it is crucial to do so here as well. Under this umbrella we find **Key Performance Indicators** (KPIs) as important as the most visited pages, the average time the user spends on each page, the number of visits, the number of users or the location from where the content is consumed. With this information we can better plan our analysis of availability, health checks, latency, etc. from an infrastructure point of view. Furthermore, we are able to take informed decisions on the content and its structure.
- **Contribution perspective**: we cannot forget that the fundamental pillar of Innersource is collaboration and transparent contribution. One of the 5 top myths about Innersource is that We'll get free work. This is radically untrue. 90% of the contributions will originate from within the team itself, which is why it is crucial to understand who is contributing, how much they are contributing, and to design a reward model for these users. This way we ensure that the community is active, motivated and quality content is generated. Regardless of the technology used to manage these contributions, KPIs such as the number of Pull Requests generated per user, the average life cycle of Pull Requests, the number of open issues, the average time it takes to close issues or the average delivery time, must be measured and monitored. This will help us to improve the community.

What

The goal of monitoring is not to measure for the sake of measuring. We need to understand what we want to deliver and ensure that monitoring provides us with the means to measure whether we are meeting our objectives.

Key Performance Indicators (KPIs) are the critical (key) indicators of progress toward an intended result. KPIs provides a focus for strategic and operational improvement, **create an analytical basis for decision making and help focus attention on what matters most**. As Peter Drucker famously said, *"What gets measured gets done."*

We will be leveraging KPIs along this section as main tool for continuously improve. It is important to note that what is offered as per evergreen delivery IP guidance is a basic and generic selection of measures that are recommended in any Innersuorce project. However, some of them may not be relevant in certain contexts, or there may be other KPIs that are not being taken into account here. Therefore, these are simple recommendations and we are open to contributions and suggestions.

Following the same scheme as in the **why** section, we can divide the KPIs according to two lenses, consumption and contribution. The possibilities are endless, but here you can find some examples:

- **Consumption lense** example KPIs would be:
    - Top consumed articles
    - Number of unique roles of the users
    - Average time spent by article
    - Number of views by article
    - ...
- **Contribution lense** example KPIs would be:
    - Total number of Pull Requets

- Current open issues number
- Total contributors
- Total number of closed issues
- ...

Note

Remember, the list of KPIs is endless. Also, these are not mandatory KPIs. Make sure you take time to understand your scenario and find the best suit of KPIs that help you govern you Innersource project.

How

First thing we need to think about is where are we getting the data from. As per evergreen delivery IP guidance, the reference architecture defines the Application Insights as the main resource for monitoring. We have chosen this resource because of how powerful it is and the amount of data it is able to log a store. In the Self Hosted Site section you can find information about how to deploy it along with all the other resources.

Application Insights would be our best ally when it comes to infrastructure metrics and usage metrics. This would cover our consumption needs. However, when coming into the contributions metrics we need to find a different solution. There are two main approaches we are considering in this case, **Azure DevOps** and **GitHub**. You can approach them from two different points:

- Leveraging **1es CloudMine Solution**. This is recommended approach. CloudMine is an engineering system designed to collect, curate and deliver data for software engineering analytics, monitoring and scorecards. CloudMine aggregates data from diverse sources including Azure DevOps, HeadTrax, Stack Overflow etc. This solution is super handy since you only need to onboard your repository, either ADO or GitHub. CloudMine already has some predefined tables with information you can simply query by using Kusto Query Language.
- Taking advantage of **public APIs**. Both, ADO and GitHub have their own APIs that you can always query if there is any missing information that you can't access via CloudMine.

Finally, the only missing piece is the representation of this data. How can we have an easy and quick overview of what has been happening? Again, there are a lot of different solutions available, you need to identify which ones better suit your team. This guidance we suggest evergreen IP repositories to leverage PowerBI dashboards and templates provided as a starting point.

What's next?

If you liked what you read and want to know how to apply it, please go to the How to Setup Infrastructure section.

How to set up the infrastructure

The core infrastructure consists of a web application whose content is deployed through Azure Pipelines and on top there is an App Insights that is responsible of monitoring all the activity that occurs (User spent time, URLs most consumed, etc).

Requirements

The following Azure services are required:

- Application service
- Application Insights
- Function App (if we have AAD data)
- Storage Account Table (if we have AAD data)
- Key vault (if we have AAD data)
- Power BI Online
- Service Principal with reader permissions on Microsoft tenant to run the Function App (if we have AAD data)

Solution Flow

Every time a user enters to the documentation page all the activity is monitored by App Insights and published into a Power BI Dashboard. We obtain the user names from each of the sessions with a little addon on the header of the pages (you can check how to configure the addon on the following section)

At this point we have two different approaches (and two different templates, one for each case): the first one with AAD data and the second one without this data.

Without AAD Data

If we don't have the proper permissions to extract user data from the tenant the information/user activity is retrieved directly from Application insights workspace and embedded to the Power BI report.

With AAD Data

There is some of the information like the user name, the job level, organization, manager name ... that needs to be extracted from the Microsoft AAD tenant because is not available in App Insights/Claims/Token. If we have AAD permissions the function app that retrieves this information using the user names as a key parameter and saves the results into a Storage account table.

Note

This function must be run with a managed identity with **User.Read.All** permissions on the tenant

This storage account table is connected to the report using a connector and this allows us to enable auto refresh. The reason why we're doing this is because currently Power BI doesn't support dynamic data sources, and if we want to retrieve information from the graph API the URL is generated using the user name dynamically.

Web App Configuration to extract the user names

App Insights does not store the name of the users accessing the web app, it stores the sessions. To relate these sessions with the user names and to be able to extract metrics, this piece of code needs to be added to the header.

Note

It needs to be updated with the corresponding instrumentation key.

These are the instructions on how to configure the header in each page to retrieve the user information (as an example we're using the modern delivery):

- Go to the file file: ModernDeliveryInnerSource/head.tmpl.partial at main · microsoft-consulting-services/ModernDeliveryInnerSource (githubenterprise.com)
- Replace/add the following code under the tag:

Code:

```
<script type="text/javascript">
!function(T,l,y){var S=T.location,u="script",k="instrumentationKey",D="ingestionendpoint",C="disableExceptionTracking",E="ai.device.",I="toLowerCase",b="crossOrigin",w="POST",e="appInsightsSDK",t=y.name||"appInsights";(y.name||T[e])&&(T[e]
src: https://az416426.vo.msecnd.net/scripts/b/ai.2.min.js,

cfg: {
+     instrumentationKey: "XXXXXXXXXXXXXXXXXXXXXX"
}});
+ var telemetryInitializer = (envelope) => {
+ envelope.data.someField = 'This item passed through my telemetry initializer';
};
+ appInsights.addTelemetryInitializer(telemetryInitializer);
+ appInsights.trackTrace({message: 'This message will use a telemetry initializer'});
+ var id = null;
+ fetch('/.auth/me').then(function(response){
+ return response.json();
+ }).then(function(data){
+ id = data[0].user_id;
+ appInsights.setAuthenticatedUserContext(id);
});
</script>
```

Function App

The function app runs Powershell code to extract the user information from the user activity on App Insights and store it into an Azure storage table. This function is triggered by time every 5 minutes and needs to be run by an identity with reader permissions on the Microsoft AD tenant.

Warning

Currently we're using a user account (instead a managed identity) with all the credentials stored in a Keyvault because the permissions were not granted yet (there's a support case ongoing to obtain the required permissions).

```
# Input bindings are passed in via param block.
param($Timer)

#############################################################################

Write-Host "Selecting subscription..."
Select-AzSubscription "77e5ad56-8b80-44a1-b5f0-f76d48aa44c9"

az account show

$appIdSecret = Get-AzKeyVaultSecret -VaultName "ccoe-kva" -Name "appId"
$appIdSS = [System.Runtime.InteropServices.Marshal]::SecureStringToBSTR($appIdSecret.SecretValue)
try {
    $appId = [System.Runtime.InteropServices.Marshal]::PtrToStringBSTR($appIdSS)
}
finally {
    [System.Runtime.InteropServices.Marshal]::ZeroFreeBSTR($appIdSS)
}

$appKeySecret = Get-AzKeyVaultSecret -VaultName "ccoe-kva" -Name "apiKey"
$appKeySS = [System.Runtime.InteropServices.Marshal]::SecureStringToBSTR($appKeySecret.SecretValue)
```

```
try {
    $apiKey = [System.Runtime.InteropServices.Marshal]::PtrToStringBSTR($appKeySS)
}
finally {
    [System.Runtime.InteropServices.Marshal]::ZeroFreeBSTR($appKeySS)
}

$Graphcred = Get-AzKeyVaultSecret -VaultName "ccoe-kva" -Name "mycred"
$CredKeySS = [System.Runtime.InteropServices.Marshal]::SecureStringToBSTR($Graphcred.SecretValue)
try {
    $CredKey = [System.Runtime.InteropServices.Marshal]::PtrToStringBSTR($CredKeySS)
}
finally {
    [System.Runtime.InteropServices.Marshal]::ZeroFreeBSTR($CredKeySS)
}

# EXAMPLE: "X-Api-Key" = "key1:GUID1,key2:GUID2"
$headers = @{"Content-Type" = "application/json" }
$headers.add("x-api-key", $apiKey)

$query = @{"query" = "pageViews | where timestamp >= ago(60d) | extend refUri_ = tostring(customDimensions.refUri) | where url contains('ccoe.azurewebsites.net') | distinct user_AuthenticatedId | where (user_AuthenticatedId contains 'microsoft.c
$body = ConvertTo-Json $query | % { [regex]::Unescape($_) }

Write-Host "Retrieving information from Application Insights..."

$result = Invoke-RestMethod "https://api.applicationinsights.io/v1/apps/$appId/query" -Headers $headers -Body $body -Method POST
$userlist = $result.tables.rows

$UsersGraph = @()

############################################################
Write-Host "Getting user data..."
az login -u XXXXXXX@microsoft.com -p $CredKey --tenant "72f988bf-86f1-41af-91ab-2d7cd011db47"
az account get-access-token --resource "00000003-0000-0000-c000-000000000000"
$AuthResponse_access_token = ""


foreach ($user in $userlist) {
    $UsersGraph += Invoke-RestMethod -Uri "https://graph.microsoft.com/v1.0/users/$($user)" -Headers @{Authorization = "Bearer $($AuthResponse_access_token)" }
}

#storage account variables
Write-Host "Accessing the Storage Account..."

$resourceGroup = "ccoe-playbook"
$storageAccountName = "storageaccountccoepbc00"
$storageAccount = Get-AzStorageAccount -Name $storageAccountName -ResourceGroupName $resourceGroup
$ctx = $storageAccount.Context
$cloudTable = (Get-AzStorageTable -context $ctx -ErrorVariable ev -ErrorAction SilentlyContinue).CloudTable
$partitionKey = "Users"

Write-Host "Table name :" $cloudTable

foreach ($user in $UsersGraph) {
    Write-Host "Processing user: $($user.userPrincipalName)"
    if ($cloudTable) {
        try {
            Add-AzTableRow -table $cloudTable -partitionKey $partitionKey -rowKey $user.id -property @{"UserDisplayName" = $user.displayName; 'JobTitle' = $user.jobTitle; 'UserPrincipalName' = $user.userPrincipalName; } -UpdateExisting
        }
        catch{
            Write-Host "User $($user) Not Found"
        }
    }
    else {
        Write-host 'Table Storage Reference Not Found!'
    }
}
Write-Host "Table updated!"
############################################################################################

# Get the current universal time in the default string format.
$currentUTCtime = (Get-Date).ToUniversalTime()

# The 'IsPastDue' property is 'true' when the current function invocation is later than scheduled.
if ($Timer.IsPastDue) {
    Write-Host "PowerShell timer is running late!"
}

# Write an information log with the current time.
Write-Host "PowerShell timer trigger function ran! TIME: $currentUTCtime"
```

Infrastructure deployment

To deploy the infrastructure we use a yaml pipeline from which we deploy a **Key vault**, **Application insights** and a **Windows App service plan** and then for each of the stages (integration, staging and prod) it will deploy/update the app service with all the changes following a **runcOnce** strategy (Cannary and Rolling are also supported).

Note

Cannary (reduce risk by slowly rolling out the change to a small subset of users. As you gain more confidence in the new version, you can begin to release it to more servers in your infrastructure and drive more users to it) and Rolling (replacing instances of the previous version of an application with instances of the new version of the application on a fixed set of machines (rolling set) in each iteration) are also supported.

Common steps

```
steps:
  - checkout: self
  - task: AzureResourceManagerTemplateDeployment@3
    displayName: 'key vault'
    inputs:
      deploymentScope: 'Resource Group'
      azureResourceManagerConnection: $(azureRMServiceConnection)
      subscriptionId: $(subscriptionId)
      action: 'Create Or Update Resource Group'
      resourceGroupName: $(resourceGroupName)
      location: $(resourceGroupLocation)
      templateLocation: 'Linked artifact'
      csmFile: '$(keyVaultTemplateFilePath)'
      csmParametersFile: '$(configFilePathKeyVault)'
      deploymentMode: 'Incremental'
      deploymentOutputs: 'OUTPUTS_KEYVAULT'
      pwsh: true
  - task: AzureResourceManagerTemplateDeployment@3
    displayName: 'application insights'
    inputs:
      deploymentScope: 'Resource Group'
      azureResourceManagerConnection: $(azureRMServiceConnection)
      subscriptionId: $(subscriptionId)
      action: 'Create Or Update Resource Group'
      resourceGroupName: $(resourceGroupName)
      location: $(resourceGroupLocation)
      templateLocation: 'Linked artifact'
      csmFile: '$(applicationInsightsTemplateFilePath)'
      csmParametersFile: '$(configFilePathApplicationInsights)'
      deploymentMode: 'Incremental'
      deploymentOutputs: 'OUTPUTS_APPLICATIONINSIGHTS'
      pwsh: true
  - task: AzureResourceManagerTemplateDeployment@3
```

```
      displayName: 'windows asp'
      inputs:
        deploymentScope: "Resource Group"
        azureResourceManagerConnection: $(azureRMServiceConnection)
        subscriptionId: $(subscriptionId)
        action: "Create Or Update Resource Group"
        resourceGroupName: "$(resourceGroupName)"
        location: $(resourceGroupLocation)
        templateLocation: "Linked artifact"
        csmFile: "$(appServicePlanWindowsTemplateFilePath)"
        csmParametersFile: "$(configFilePathAppServicePlanWindows)"
        deploymentMode: "Incremental"
        deploymentOutputs: "OUTPUTS_APPSERVICEPLANWINDOWS"
```

Stage/environment steps:

```
- deployment: int
  displayName: "deploy windows webapp"
  pool:
    vmImage: $(vmImageName)
  environment: $(intEnvironmentName)
  strategy:
    runOnce:
      deploy:
        steps:
          - checkout: self
          - task: AzureResourceManagerTemplateDeployment@3
            displayName: 'app service'
            inputs:
              deploymentScope: 'Resource Group'
              azureResourceManagerConnection: $(azureRMServiceConnection)
              subscriptionId: $(subscriptionId)
              action: 'Create Or Update Resource Group'
              resourceGroupName: $(resourceGroupName)
              location: $(resourceGroupLocation)
              templateLocation: 'Linked artifact'
              csmFile: '$(appServiceWindowsTemplateFilePath)'
              csmParametersFile: '$(intConfigFilePathAppServiceWindows)'
              deploymentMode: 'Incremental'
              deploymentOutputs: 'OUTPUTS_WEBAPP_INT'
```

Once everything is deployed you can check the results on the different URLs:

- Integration
- Staging
- Production

Telemetry pipeline

For the telemetry infrastructure we're deploying a function app and a new service plan to host it (the function app was described previously). We have the **deploy.yml** and **variables.yml** files under the /pipelines/telemetry/ folder, the **application-insights.json** and **function-app.json** under pipelines/iacs/ folder and the function parameters **egipfapp-telemetry.json** and **egipsp-telemetry.json** under /platform/telemetry/ folder.

Variables defined

```
## Pipeline config
# The ADO Service Connection to be used for the deployment
azureRMServiceConnection: "Evergreen-Delivery-IP"
#T he subscription ID where Infrastructure will be deployed
subscriptionId: "4d186e10-e0a0-4555-a97f-7c8c2272d741"
# Agent VM image name
vmImageName: "ubuntu-latest"
# View pipeline logs
System.Debug: true
# Path to platform IaC configuration files
configFilesPath: "$(Build.SourcesDirectory)/platform"
# Path to IaC modules path
iacsTemplatesPath: "$(Build.SourcesDirectory)/pipelines/iacs"
# Path to Windows App Service Plan module deployment template file
appServicePlanWindowsTemplateFilePath: "$(iacsTemplatesPath)/app-service-plan-windows.json"
# Path to Windows Function App module deployment template file
functionAppTemplateFilePath: "$(iacsTemplatesPath)/function-app.json"

## Environment config
# Name of the Resource Group where the infrastructure will be deployed
resourceGroupName: "Evergreen-Delivery-IP"
# Location of the Resource Group where the infrastructure will be deployed
resourceGroupLocation: "West Europe"

## Common
# Name given to the Azure pipelines deployment environment with deployment steps for the common hosting infrastructure elements, listed below
telemetryEnvironmentName: "$(resourceGroupName)-telemetry"
# Path to the Windows App Service Plan configuration file used to host the App Services for all environments
configFilePathAppServicePlanWindows: "$(configFilesPath)/telemetry/egipsp-telemetry.json"
# Path to the Windows App Service configuration file used to host the web app for the integration environment
telemetryConfigFilePathFunctionApp: "$(configFilesPath)/telemetry/egipfapp-telemetry.json"
```

Deployment steps

```
    deploy:
      steps:
        - checkout: self
        # deploy windows app service plan
        - task: AzureResourceManagerTemplateDeployment@3
          displayName: "windows asp"
          inputs:
            deploymentScope: "Resource Group"
            azureResourceManagerConnection: $(azureRMServiceConnection)
            subscriptionId: $(subscriptionId)
            action: "Create Or Update Resource Group"
            resourceGroupName: "$(resourceGroupName)"
            location: $(resourceGroupLocation)
            templateLocation: "Linked artifact"
            csmFile: "$(appServicePlanWindowsTemplateFilePath)"
            csmParametersFile: "$(configFilePathAppServicePlanWindows)"
            deploymentMode: "Incremental"
            deploymentOutputs: "OUTPUTS_APPSERVICEPLANWINDOWS"

        - task: AzureResourceManagerTemplateDeployment@3
          displayName: "function app"
          inputs:
            deploymentScope: "Resource Group"
            azureResourceManagerConnection: $(azureRMServiceConnection)
            subscriptionId: $(subscriptionId)
            action: "Create Or Update Resource Group"
            resourceGroupName: "$(resourceGroupName)"
            location: $(resourceGroupLocation)
            templateLocation: "Linked artifact"
            csmFile: "$(functionAppTemplateFilePath)"
            csmParametersFile: "$(telemetryConfigFilePathFunctionApp)"
            deploymentMode: "Incremental"
            deploymentOutputs: "OUTPUTS_APPSERVICEPLANWINDOWS"
```

Power BI suggested templates

As introduced in previous sections, as per evergreen delivery IP guidance is to use the Power BI tool to build monitoring dashboards. Before starting to explain each of the templates, it is important to know that Power BI consists of several elements that work together, starting with these three basic concepts:

- Windows desktop application called Power BI Desktop.
- Online SaaS (software as a service) service called Power BI service.
- Power BI mobile applications for Windows, iOS and Android devices.

As part of this IP guidance we will only refer to the first two, desktop application and SaaS service. These will be the two services we will use for our monitoring solution.

In other sections we have already discussed the two ambits to consider for monitoring: the consumption and the contribution. To ensure that the dashboards are modular and can easily evolve as new metrics and KPIs are identified, it is proposed to dedicate a dashboard for each type of service. On the other hand, for the contribution type of monitoring, since this IP refers to two types of technologies, GitHub and ADO, both have been differentiated. The reasoning behind is that the data model of each one is not exactly the same. Thus, in the next articles we will find more detailed information about:

Note

The following templates are built upon a set of metrics selected by the Evergreen delivery IP team based on the experienced with other teams from Microsoft. These KPIs have been identified as the most useful ones when monitoring innersource type of resources. However, they are just guidelines and rails from where teams can increment and customize their solutions according to their specific requirements.

- **Documentation lens consumption insights template**: this template provides a holistic view of the application's consumption. Among its metrics we find important data such as the number of **views**, the number of unique **users**, the **most consumed articles** or the average **time that each user spends** on each article.

Warning

The following two templates are still under development. We stimate to have them available by the end of Q1FY22. The following images are just POCs and sketchs of the content of these dashboards that may be updated in the upcoming months.

- **GitHub contribution insights template**: a GitHub flavored type of template for those repositories that are hosted by GitHub. KPIs of this dashboard include total number of **contributors**, amounts of **additions and deletions** by month, **Pull Requests lifecycle**, etc.

- **Azure DevOps contribution insights template**: an Azure DevOps flavored type of template for those repositories that are hosted by ADO. This template takes advantage of the Continuous Planning capabilities implemented in ADO to give a more broad overview of the contributions and planned work.

Note

Everything in these templates can be customized as per wanted. Backgrounds, field names, field formats and more can be adapted to the requirements and needs of your specific scenario.

What's next?

Start implementing your documentation lensed dashboard by following the instructions in Documentation lens insights template section.

Documentation lens consumption insights template

This template seeks to collect as much information as possible about the consumption of the documentation lens website. Understanding how users navigate through the pages, what content they consume most frequently or how the website grows in terms of consumption over time allows informed decisions to be made about the content and form with which the platform should evolve. As already introduced in other sections, this template is fed by the Application Insights against which the Web App is connected. This resource will be our source of the data model.

One of the most important data identified by the evergreen delivery IP team is **who** is accessing the content. Since the content to be pulished by innersource is internal, this guide shows how to set up authentication/authorization as part of the infrastructure. This configuration, together with other specifications defined in the how to setup infrastructure section, allows retrieving the UPN (User Principal Name) of the user accessing the content. Through the UPN and by making REST calls to the AzureAD API, we can know data such as the user's role, location and manager. Depending on the content that is being innersourced, this can be fundamental data when determining the impact and potential opportunities with customers that can be generated.

However, there are certain limitations in the Power BI service when it comes to making dynamic calls against any URL. The content that is published in the Power BI SaaS service cannot be dynamic content. This means that there can be no data sources that depend on other sources. For example, if we make a call to App Insights to retrieve the user's username and then use this data to gather more information from the Azure Active Directory, the Power BI web publishing service will detect a dependency between data sources and will not allow to auto-refresh the dashboard. Therefore, two different templates are available:

- **AAD integrated template**: this template implements calls against the Graph API and includes information such as how many different roles consume the content, where the user is based or to which organization the user belongs. This is the most information-rich option but cannot be published in the Power BI SaaS service.

- **No AAD integrated template**: for those scenarios where having the dashboard published and accessible through the internet is a requirement, this template allows you to do so although, as a counterpoint, it does not provide user information beyond the UPN.

How to configure the AAD integrated template

This template relies on the infrastructure explained in the With AAD Data section. To setup your dashboard by leveraging the AAD integrated template, please follow these steps:

1. If you don't have it already download and intall Power BI Desktop application from the Microsoft Store
2. Find the AAD integrated template here and download it.
3. Before opening the template, we first need to clear the credentials. Open Power BI Desktop application. Go to File > Options and settings > Data source settings. From the Global permission tab, select all sources showing and click on Clear Permissions. After that close the window.

Note

If you don't have any permissions pre-configured in your Power BI Desktop, then you are good to go.

1. Open the template by going to File > Open report > Browse report. While loading, it will ask you for some data. Please follow the instructions given in the description boxes and fill in the Application insights workspace ID and the storage account table URL.

   As shown in the video, for the Application Insights connection, you will be asked to connect with your credentials. Please, when promped for your credentials, select Organization Account and enter your Microsoft account. Once signed in, click connect.

   For the storage account connection, you will also be prompted for the Account key of such Storage. Please, go Azure portal > Storage Account > Select your Storage Account > Access Keys > key 1. Copy this value and paste it in the box. Then click connect.

Now you are ready to start monitoring the consumption of your innersource application!

How to configure and publish the No AAD integrated template

This template relies on the infrastructure explained in the Without AAD Data section. Please, use this template if you want it to

be published in the Power BI SaaS service. To setup your dashboard please follow these steps:

1. If you don't have it already download and intall Power BI Desktop application from the Microsoft Store
2. Find the No AAD integrated template here and download it.

3. Before opening the template, we first need to clear the credentials. Open Power BI Desktop application. Go to File > Options and settings > Data source settings. From the Global permission tab, select all sources showing and click on Clear Permissions. After that close the window.

Note

If you don't have any permissions pre-configured in your Power BI Desktop, then you are good to go.

1. Open the template by going to File > Open report > Browse report. While loading, it will ask you for some data. Please follow the instructions given in the description boxes and fill in the Application insights workspace ID.

   As shown in the video, for the Application Insights connection, you will be asked to connect with your credentials. Please, when promped for your credentials, select Organization Account and enter your Microsoft account. Once signed in, click connect.

Now you are ready to start monitoring the consumption of your innersource application!

Behind the scenes: the data model

In this section we will explain what the data model is and how to work with it. This is important since there may be a scenario in which the template does not sufficiently fit the level of detail your team is looking for/needs.

In order to see which tables are being used, what fields each table has and how they relate to each other, click on the Model button in Power BI Desktop. This view gives you, at a glance, all the relevant information from which the various charts are then fed. Use it to find out if the new fields you need are already available, or if you should alter the model itself to fit your scenario.

If the format of the data that appears is not the right one, or you need other data, then you will have to modify the queries that are made to the different data sources. To do this, click on Transform data. A new window will open where, by clicking on Advanced editor, you will be able to see the original call together with all the transformations made to the data. In addition, from here you can also edit the template input parameters. Once you have the data to your liking, click on Close & Apply to close this window and load the new tables in your model.

<Your> Lens

This is a placeholder section for any other lens that needs to be added to evergreen delivery IP to help in managing such types of IP assets. Please feel free to provide feedback or contribute with your ideas by submitting a PR.

Resources

This section fo the site provides description and references to other approaches, teams, external tools that the initiative depends on or leverages. This informaiton can be used as reference for getting more insights on the approach defined.

The main resources are details below :

- innersource introduction and links to relevant learning external and internal learning materials
- knowledge management tools relied on for the implementation of the approach
- examples of teams adopting Evergreen Delivery IP (and contributing approaches, tools and guidance)

Innersource

The evergreen delivery initiative is promoting the delivery IP development and sustaining over time using the innersource collaboration approaches.

What is Innersource?

In short innersource is use of open source development principles inside of the company enabling limited audience of company employees to view, reuse, submit issues and over time start contributing to the repositories of IP.

Learn More

Innersource commons organization has comprehensive set of learning materials introducing innersource and discussing the patterns and anti-patterns captured in companies that have implemented innersource principles.

Specifically:

- Introduction to Innersource - learning path providing good introduction into innersource
- Details behind the innersource roles:
  - Trusted Committer - information for duties and techniques that core IP team members would need to follow
  - Contributor - information on contributor duties as part of an innersource projects. Note that contributions can be for different types of assets and therefore of different time consumption.

Innersource in Broader Microsoft

Innersource is being adopted in a wider Microsoft as well. See this good overview of how Microsoft product teams have adopted the innersource over last couple of years, what are the learnings:

Tools Used

List of tools and approaches that Evergreen Delivery IP depends on are:

- Project Chrysalis - metadata based tool for gathering new ideas and also being able to find innersource repositories, assets they produce
- SEE - set of tools integrated into the project pipeline of the Services supporting:
  - Creation of new Teams sites for opportunities and projects
  - Promoting IP relevant to the projects, getting insights into use of the IP
  - Enabling feedback on IP used in projects
- 1ES - overall Microsoft engineering system that also drive the policies around our internal ADO and GitHub Enterprise repositories
- Visual Studio Code - swiss army knife of tools that can be used to access the content in your repositories, author well defined Markdown based documentations

Example Teams

Example teams following the approach and guidance materials of hte evergreen delivery IP are:

- Evergreen Delivery IP team itself is using the evergreen delivery IP approaches and given its innersource (open internally) nature can be used as example of how a team using these approaches would set up their MS Teams team and also ADO based Git repo.
- DevOps DoJo team is contributing a lot of best practices and tools from their specific IP management approaches and learnings from innersource pilot
    - Innersource repository for those that have been provisioned on GitHub Enterprise instance of Microsoft