Table of Contents

Hands-on Labs Content Package Development

The Hands-on Labs platform makes it as easy as possible to bring your lab or demo content to Microsoft customers. If you have experience with standard technologies such as Azure Resource Manager (ARM), PowerShell, Azure and Markdown, you have all the skills required to produce content for Hands-on Labs.

Content packages use a hierarchical arrangement of Tracks, Experiences, and Stories. Stories are sets of instructions and a custom environment in which the customer can follow those instructions. The platform supports three types of story deployment:

1. Self-paced Gallery Lab: available from the Hands-on Labs website as on-demand content for Microsoft customers.

2. Facilitator-led sessions: booked ahead of time by field and partners. In these sessions, related stories, collected into Experiences, are presented to groups.

3. Demos: available from Hands-on Demos website as an on-demand content for Microsoft customers and Field

Your content may not support both forms of presentation, and you control how it is made available within the platform.

Building a Content Package

Organize content packages with the following hierarchy:

1. **Track**

    - Each track has a Title and Description.
    - A track contains a number of related experiences.
    - Tracks are selected first when booking a Facilitator-led session.

2. **Experience**

    - Each experience has a Title and Description.
    - An experience contains a number of related stories.
    - Experiences grouped into a track must have the same base jump host definition. Stories within an experience can customize the resources available by providing additional ARM templates.

3. **Story**

    - Each story has a Title and Description, as well as lots of other metadata for filtering purposes.
    - Each story may also include an ARM template for deploying resources and assets.

- Individual stories can optionally appear in the Self-Paced Gallery as stand-alone labs. These stories do not have to be available as a Facilitator-led experience. However, they may be made available that way if desired.

Creating a Story

There are two key steps to consider when authoring a new story:

1. Instructions: a set of Markdown files that include the step-by-step instructions for users.

2. Deployment: an ARM template that describes the resources required to run your content.

Once you have these, you are ready to onboard. This involves a quick review to ensure the lab style conforms to the "Hands-on Labs Style Guide", content aligns with Microsoft standards and that deployment templates are reliable and cost-efficient.

Instructions

Story instructions are written in Markdown. These scenarios are presented to the user via the Hands-on Labs Guide app. Each story has at least three Markdown files:

1. Introduction: a brief introduction to the story. Typically this content sets the scene including the business scenario and technologies used.

2. Scenarios: one or more numbered pages that contain the step-by-step lab instructions.

3. Conclusion: a brief conclusion to the story. Typically this recaps what the user has seen and highlights the key takeaways.

Deployment

Each story must have a single ARM template that describes all of the resources the user needs for the session. Within the ARM template, the story will provision a "Jump Host" virtual machine. This VM will host the Guide app and any other applications the user will need to complete the story.

Every Hands-on Labs session is deployed into a new Azure Resource Group. A temporary AAD user account is generated for the customer and, by default, granted read-only access to the group. All resources deployed for the session are deployed into this resource group and no resources are shared with any other sessions †. When the session is over, the resource group, AAD user, and any resources created during the lab are automatically deleted.

Permissions

In the interest of security, the default behavior is to provide read-only access to the resource group. You may use PowerShell to modify the role-based access of the user if your lab requires additional permissions. However, you must use the minimal set of roles

required to create the resources for the lab. The full list of available roles and the actions allowed to each can be retrieved from Azure with the PowerShell command `Get-AzureRmRoleDefinition`. See Appendix B for the suggested subset of roles recommended for Hands-on Labs.

Valorem will reject scripts that elevate the user to "Contributor" or higher as these roles pose a significant risk to the platform.

† Except for some limited cases in facilitator-led sessions

Important: users get *access* to a resource group, not a subscription. This means the old portal is not available.

Next steps

The remainder of this document gives a detailed overview of the Hands-on Labs Content Platform and the tools we have available for content development.

Hands-on Labs Content Repository

All Hands-on Labs content packages are stored in a *Hands-on Labs content repository*. This is a Git version control system with `.json` metadata files, `.md` Markdown files, and any scripts and ARM templates required. Large assets and custom VHD images are stored directly in Azure Storage, and not in the content repository. Additional information regarding the use of Git is available in the "Hands-on Labs Content Production and Publication" document.

Repository layout

The Hands-on Labs content repository is made up of a number of configuration files and loose files, which are parsed and packaged up by the **content processor**.

Each of the configuration files must be placed in its own directory. The path from the root of the content repository to a given configuration file is used to construct the **ID** of that content. A Hands-on Labs content ID is in URI format, with the scheme identifying the type of the content.

For example, given a path to a `story.json` config file "`~/single/experience1/story1`", the story ID might be "`story://content-repo/single/experience1/story1`".

Use a hierarchical layout for content to ensure Experiences are presented in the specified order by the Guide app. For more information on naming conventions, refer to the "Hands-on Labs Content Package Naming Conventions" guide.

Here are examples of both single track and multiple track layouts:

```
Single Track Example

root/
    repoconfig.json
```

```
    single/
        track.json
        experience1/
            experience.json
            story1/
                content/
                deployment/
                content.json
            story2/
                content/
                deployment/
                content.json
        experience2/
            experience.json
            story1/
                content/
                deployment/
                content.json
            story2/
                content/
                deployment/
                content.json
```

This will result in the following set of content IDs:

```
     track://root/single
experience://root/single/experience1
     story://root/single/experience1/story1
     story://root/single/experience1/story2
experience://root/single/experience2
     story://root/single/experience2/story1
     story://root/single/experience2/story2
```

Multi Track Example

```
root/
    repoconfig.json
    multi/
         datamanagement/
            track.json
            experience1/
                experience.json
                story1/
                    content/
                    deployment/
                    content.json
                story2/
                    content/
                    deployment/
```

```
                    content.json
            experience2/
                experience.json
                story1/
                    content/
                    deployment/
                    content.json
        security/
            track.json
            experience1/
                experience.json
                story1/
                    content/
                    deployment/
                    content.json
                story2/
                    content/
                    deployment/
                    content.json
```

This will result in the following set of content IDs:

```
     track://root/multi/datamanagement
experience://root/multi/datamanagement/experience1
     story://root/multi/datamanagement/experience1/story1
     story://root/multi/datamanagement/experience1/story2
experience://root/multi/datamanagement/experience2
     story://root/multi/datamanagement/experience2/story1
     story://root/multi/datamanagement/experience2/story2

     track://root/multi/security
experience://root/multi/security/experience1
     story://root/multi/security/experience1/story1
     story://root/multi/security/experience1/story2
```

Repository metadata files

Hands-on Labs content packages have a number of metadata files informing the platform how to present the labs:

- repoconfig.json
- track.json
- experience.json
- content.json

Each of these files is validated by a JSON schema, which can also be used to provide IntelliSense/AutoComplete assistance in some text editors (we recommend VS Code).

Read on for more details about each of the metadata files. See Appendix D for the complete list of schema URIs.

repoconfig.json

`repoconfig.json` tracks the "name" of this content repository. This file is required, as the repository name is injected into story, experience and track IDs. This file is provided by the platform, and is not managed by the content creator. For example:

```
{
    "$schema": "https://immtest-web.azurewebsites.net/schemas/0.0.1/ImmersionRepoMetadata.json",
    "name": "root"
}
```

track.json

`track.json` contains the title, description, permissions, and list of experiences which make up a track. Optionally, you can include a category and sort_index. For example:

```
{
    "$schema": "https://immtest-web.azurewebsites.net/schemas/0.0.1/ImmersionTrackMetadata.json",
    "title": "Single Track Facilitated Lab",
    "description": "Lorem ipsum dorem, dolor sit amet",
    "experience_ids": [
        "experience://root/single/experience1",
        "experience://root/single/experience2",
    ],
    "permissions": [ "admins" ]
}
```

experience.json

`experience.json` contains the title, description, permissions, and list of stories which make up an experience. For example:

```
{
    "$schema": "https://immtest-web.azurewebsites.net/schemas/0.0.1/ImmersionExperienceMetadata.json",
    "title": "Experience No. 1",
    "description": "Lorem ipsum dorem, dolor sit amet",
    "permissions": [ "admins" ],
    "story_ids": [
        "story://root/single/experience1/story1",
        "story://root/single/experience1/story2",
    ]
}
```

content.json

`content.json` tracks metadata for a story. The metadata includes filtering criteria, time required to complete the lab, and deployment criteria including the following:

Story metadata

1. `title` - The title of this story.
2. `description` - The description of this story.
3. `permissions` - Array of permissions for groups which should have access to this story in the Gallery.
4. `regions` - Azure regions that the story can be deployed in. Used for filtering on the website.
5. `topics` - Topics this story covers. Used for filtering on the website.
6. `products` - Products this story covers. Used for filtering on the website.
7. `jobroles` - Job roles which may be interested in this product. Used for filtering on the website.
8. `level` - Technical level of the story. Used for filtering on the website.
9. `date_created` - Format YYYY-MM-DD. The date this story was created. Used for sorting in the Gallery.
10. `duration_minutes` - The average duration (in minutes) it takes to complete this story.

Content metadata

11. `persona_role` - The default 'Persona' role to use for pages/scenarios in this story. This persona can be overridden on a per-page basis.
12. `resources` - An object mapping string names to 'resources' which may be invoked by the Guide app at runtime. Each "resource" is described by a JSON object which holds configuration for one of a number of 'launch helpers'. See the **Launch helpers** section for more information.

Deployment metadata

13. `primary_template` - Path to the primary ARM template within the deployment directory. Optional, defaults to 'template.json'.
14. `depends_on` - The Story IDs of any stories on which this story depends for deployment artefacts. E.g. if Story1 deploys resources on which Story2 depends, the Story1 ID should be added to Story2's `depends_on` array.
15. `parameters` - Table of 'parameter sets' each identified by key. See below for more information.
16. `requires_licenses` - An array of licenses to apply to the Azure AD user. Currently only supports the value "powerbi".
17. `requires_office365_trial` - If set, requests an O365 trial to be provisioned via the 3Sharp API.

18. `requires_long_running_script_sentinel` - Indicates that this story has a long-running script which is not managed by the ARM deployment (e.g. a PowerShell script running on the JumpHost).

    When set, this flag causes the deployment process to halt until a file with a specific name (the "sentinel") is created in the "assets" container in the primary storage account. The long running script must create this file when it completes. If the file is not created within a specific period of time (currently three hours) then the deployment is considered "failed".

Story Resources

Resources are available for launching applications from within Hands-on Labs story content hosted in the Guide app. For example:

```
{
    "ie_google": {
        "helper": "launchie",
        "arguments": [ "http://www.google.com" ]
    },
    "notepad_example_txt": {
        "helper": "runcommand",
        "command": "C:\\Windows\\System32\\notepad.exe",
        "arguments": [ "%temp%\\example.txt" ]
    }
}
```

Launch helpers

In this example, we defined two "launch helpers", `ie_google` and `notepad_example_txt`. Each describes a configuration for a well-known "launcher" defined by the `helper` property.

We currently support the following helpers (listed with their configuration options). Required configuration is denoted with a *:

1. `runcommand` - Runs the given command using the Windows `CreateProcess` function.
    - `command*` - The path/name of an executable or command to invoke. Supports environment variable expansion.
    - `arguments` - An array of arguments to pass to the program identified by `command`. Supports environment variable expansion.
2. `launchie` - Launches Internet Explorer.
    - `arguments*` - Additional arguments to pass to the IE executable (generally the URL to browse to).
3. `launchssms` - Launches SQL Server Management Studio. May optionally attempt to automate filling out the "Connection Info" dialog.

- `server` - The "server" string to fill out.
- `authentication_mode` - If set to `sql`, select SQL auth. If set to `windows` select Windows auth.
- `username` - The "username" string to fill out (used with `authentication_mode = sql`)
- `password` - The "password" string to fill out (used with `authentication_mode = sql`)

4. `launchexplorer` - Launches a Windows Explorer window open to the given path.

    - `path*` - The path to browse to. Supports environment variable expansion.

5. `launchvisualstudio` - Launches Visual Studio with the given parameters.

    - `solution_file` - Path to the Visual Studio Solution (.sln) or Project (.csproj) to open. Supports environment variable expansion.
    - `reset_to` - Flag to indicate if the `/ResetSettings` flag should be passed to Visual Studio. With 'GeneralDevelopmentProfile' the 'General' profile is passed in argument.
    - `title_pattern` - If Visual Studio is already running, this regular expression is used to select correct window to bring into focus.

6. `launchmmc` - Launches the MMC snap-in host.

    - `arguments*` - Arguments for the MMC host. E.g. the path to the MMC snap-in to launch. Supports environment variable expansion.
    - `title_pattern` - If the MMC host is already running, this regular expression is used to select correct window to bring into focus.

7. `launchmstsc` - Launches the Windows RDP client (mstsc) with special configuration to ensure it fits in the available desktop space.

    - `connection_file` - Path to an RDP connection file to use
    - `server` - Remote RDP server to connect to
    - `window_mode` - Configure the preferred window layout: `None`, `FullScreen`, `FitToWorkspace`
    - `prompt_for_credentials` - Always prompt for credentials (/prompt)
    - `suppress_consent_prompt` - Suppress the consent prompt (/noConsentPrompt)

Story content

Each story **must** have a `content` directory. This folder contains:

Markdown pages/Scenarios

Markdown is a human and machine-readable plain text format for writing structured documents.

1. `intro.md` - (required) An introduction to the content presented in the lab.
2. `conclusion.md` - (required) A summary of the lab content and conclusion statement.
3. Pages numbered in order. For example:
    - `0.part 1.md`
    - `2.md`
    - `3.md`
    - `4.part 4.md`

Each page appears in the Guide app as a single Scenario. Page file names must start with a numeric value or the file will be skipped during content processing. A warning will be logged to the content processor output for incorrectly named files.

Other content

All other loose files are copied into the content package without modification. For example:

```
content\
    intro.md
    0.md
    1.md
    conclusion.md
    img\
        happycat.png
        sadcat.png
```

From within the markdown files, you can refer to the other content by relative path. E.g. from the markdown file `0.md`:

```
Lorem ipsum dorem ![dolor sit](img/happycat.png "A happy cat") amet.
```

Extended Markdown features

We provide a few extensions for markdown content.

Page metadata

You can **optionally** add a page metadata block to the beginning of any markdown file. This allows you to override the **title** or **persona-role** for an individual page. For example:

```
<page title="My custom title" persona-role="CIO" />

Lorem ipsum dorem, dolor sit amet.
```

If you omit the page "title" property, then a title will be extracted from the filename. For example:

1. `0.Hello, world.md` -> title "Hello, world"
2. `1.Goodbye, moon.md` -> title "Goodbye, moon"

3. `2.md` -> no title

If there is no page title metadata and no title in the filename, then a title like "Scenario #" will be generated, where # is the page number within the story.

If you omit the "persona-role" property, then the story's default Persona will be used.

Facilitator notes

Surround facilitator-only notes in a `<notes>` block. For example:

```
Lorem ipsum dorem <notes>for the facilitor's eyes only</notes>, dolor sit et.
```

These notes will be automatically stripped from "student" deployments at runtime.

Inject placeholders

You can inject values from the ARM outputs of your story using `<inject>`. The following deployment configuration values are always available to be injected:

The name of the O365 tenant (available only if the `requires_office365_trial` flag is set):

- `O365TenantName`

Details about the Azure AD user created during deployment:

- `AzureAdUsername`
- `AzureAdUserPassword`
- `AzureAdUserEmail`

You can also inject any ARM template output in your deployment dependency chain by providing the output's name as a `key` and optionally the source story ID (if omitted, defaults to the current story). There is no special syntax for injecting a Facilitator deployment output. Instead, student deployment outputs are searched first and facilitator outputs second.

For example, if you have the following `outputs` section in your ARM template:

```
"outputs": {
    "value_from_own_arm_outputs": {
        "type": "string",
        "value": "[concat('foo ', parameters('bar'), ' baz')]"
    }
}
```

And the following Markdown:

```
Lorem '<inject key="AzureAdUsername" />' ipsum dorem.

Lorem '<inject key="value_from_own_arm_outputs">default value</inject>' ipsum
dorem.
```

```
Lorem '<inject story-id="story://some/dependency/story/id" key="value_from_de
pendency_arm_outputs" />' ipsum dorem.
```

The output might be:

```
Lorem 'test123456@cloudplatimmersionlabs.onmicrosoft.com' ipsum dorem.

Lorem 'foo bar baz' ipsum dorem.

Lorem 'value from depencency outputs' ipsum dorem.
```

A final note about inject placeholders: Inject placeholders will not work when placed in **code blocks** (inline, indent or code-fence style) or **hyperlinks**.

Anchors

Ordinary anchors are automatically launched in an external web browser. For example:

```
Lorem ipsum [google.com](http://www.google.com "Google") dolor sit amet.
```

The Guide app supports a number of "Launch helpers", which allow you to define external applications which the user can launch by clicking on the hyperlink. Launch helper anchors use the `launch:` scheme. For example:

```
Lorem ipsum [Launch NOTEPAD](launch://open_foo_txt_in_notepad) dolor sit et.
```

The second part of the `launch://` URI names some *resource* defined in the `resources` section of `content.json`. You can read about the available launch helpers in the **Launch Helpers** section.

Story deployment

Each story may optionally have a "deployment" directory. This directory contains:

1. The primary ARM template
2. Any secondary ARM templates
3. An ARM parameters file for "Facilitator" deployments
4. An ARM parameters file for "Student" deployments (optional)
5. An ARM parameters file for "StandAlone" deployments (required to appear in the Gallery)
6. Other deployment assets: PowerShell scripts, etc

The `content.json` manifest must include references to the primary template and parameters files to support the deployment scenario.

Facilitator-led Session Deployment

```
{
    "primary_template": "template.json",
    "parameters": {
```

```
        "facilitator": "facilitator.parameters.json",
        "student": "student.parameters.json"
    }
}
```

For stories to appear in the Self-Paced Gallery, you must specify the "standalone" parameter set, and provide at least one Gallery permission in the `content.json`:

Self-Paced Gallery Deployment

```
{
    "permission": [ "everybody" ],
    "primary_template": "template.json",
    "parameters": {
        "standalone": "standalone.parameters.json"
    }
}
```

Combined Deployment: both Facilitator-led and Self-Paced Gallery

```
{
    "permission": [ "everybody" ],
    "primary_template": "template.json",
    "parameters": {
        "facilitator": "facilitator.parameters.json",
        "student": "student.parameters.json",
        "standalone": "standalone.parameters.json"
    }
}
```

Dependencies

Stories can *depend on* other stories for deployment. This will be a common use-case. For example:

1. Story A deploys the Jump Host
2. Story B has no ARM template, so it depends on Story A to deploy the Jump Host

Story A's content.json

```
{
    "primary_template": "deploy_jump_host.json"

    // ...other properties...
}
```

Story B's content.json

```
{
    "depends_on": [
```

```
        "story://content-private/story_a"
    ]

    // ...other properties...
}
```

A story may depend on multiple other stories if necessary.

The deployment engine automatically determines the correct order in which to deploy these stories so that all resources are available as needed. Loops in the dependency chain are errors. The Content Processor will detect and report them as such.

You may inject values from a story's dependencies' ARM templates into the inputs of that story's ARM template. See the **Parameters files** section below.

Parameters files

The Hands-on Labs deployment engine applies a preprocessing step to parameters files before invoking the ARM deployment engine. The following preprocessing hooks are available:

Config

We can inject per-deployment configuration values:

Details about the Azure AD user created by the deployment process.

- `AzureAdUsername`
- `AzureAdPassword`
- `AzureAdEmail`

Configuration for the storage account where VHDs, Guide content, etc. reside.

- `PrimaryStorageAccountName`
- `PrimaryStorageAccountKey`

Details about the JumpHost virtual machine.

- `JumpHostDnsName`
- `JumpHostUsername`
- `JumpHostPassword`

Details about the "script sentinel" file. This value is only available if `requires_long_running_script_sentinel` is set in `config.json`.

- `ScriptSentinelFileName`

To inject one of these config values into an ARM template use the `config:` prefix. For example:

```
{ "value": "config:PrimaryStorageAccountName" }
```

Is transformed into:

```
{ "value": "imm0ad11gr8h6jy5d7w51" }
```

Assets

We can inject blob URIs for any asset in the deployment directory. This simplifies the task of deploying files required during deployment, e.g. PowerShell scripts for preparing a VM. Use the `asset:` prefix along with the path to a deployment directory file.

E.g. given the file `deployment/scripts/ConfigureAD.ps1` in the deployment directory, the following configuration will cause it to be copied into the deployment package. At deployment time, this file will be uploaded to blob storage and a fully-qualified read-only blob storage URI will be injected into the ARM template.

```
{ "value": "asset:scripts/ConfigureAD.ps1" }
```

Is transformed into:

```
{ "value": "https://immdeployment.blob.core.windows.net/deployment/Content-Private-ds48g6h1t2/story-id/deployment/scripts/ConfigureAD.ps1?sv=..." }
```

Blobs

We can inject blob URIs for blobs stored in the Hands-on Labs Storage service. This information determines which blobs to copy during the *Blob Copy* deployment step.

When a lab is deployed to the platform, the parameters files are scanned for `blob:` values like the one below. These are resolved to a blob in the Storage Service, and the blob is copied into the user's Blob Storage account.

All blob content not explicitly included in the content package should be deployed using this mechanism. This includes VHDs, data files, and any other payload on which the content depends that is not checked in to the repository.

```
{ "value": "blob:dii/20160616/Win10JumpHost.vhd" }
```

Is transformed into:

```
{ "value": "https://sa031568.blob.core.windows.net/assets/20160616/Win10JumpHost.vhd" }
```

ARM template outputs from dependencies or the facilitator deployment.

To inject a value from the outputs of the ARM templates of a story's dependencies, use *output resolution syntax*. You must provide the story ID, output name and the expected "source".

The available "sources" are:

1. `facilitator` - Search the ARM outputs of the Facilitator deployment. Only available for Student deployments within a Facilitator-led Session deployment.

2. `dependency` - Search the ARM outputs of your own dependencies. Always available to stories with dependencies.

For example:

```
{
    "story1_sql_server_dns_name": {
        "value": {
            "source": "dependency",
            "story_id": "story://root/dii/experience1/story1",
            "output": "sql_server_dns_name"
        }
    }
}
```

Is transformed into:

```
{
    "story1_sql_server_dns_name": {
        "value": "sql01.contoso.com"
    }
}
```

Content processor

You can verify your content using the Content Processor in the Hands-on Labs Content Vendor Tool. This tool performs static content checking as it process content into packages consumable by the Guide app. The tool prints log messages as the content is processed, including any errors encountered.

The Content Processor is also part of the content verification process performed after a Pull Request is created in the Content Repository. The vendor must validate all content before submitting it for review.

The Hands-on Labs Guide app

The Guide app runs on the JumpHost, where it presents Hands-on Labs scenarios to the user.

Including the Guide app in the Content Package

Installation of the Immersion PowerShell module and Guide app is a prerequisite for content to be displayed. This is accomplished by including the Microsoft Custom Script Extension in the ARM template for your JumpHost virtual machine with instructions similar to the example template provided in the Content repository under $/../platform/example/experience1/3_hol_only/deployment/install_guide_app.json.

Including the “installGuide” property in the extension's runtime settings array and setting it to true will ensure the Guide app is installed on the JumpHost and made ready for the end-user.

Deployment process

At the end of the lab deployment, all of the content packages required by the Guide are copied into a storage account container named `jumphost` in the lab’s resource group. The deployment process then creates the Guide configuration file `config.json` in the same container (see below for more info).

JumpHost preparation

After downloading and extracting the Guide app to the JumpHost, the JumpHost bootstrap script must finally generate the `app.startup.json` configuration the Guide app uses to connect to the primary storage account and retrieve both `config.json` and Hands-on Labs content packages.

The Guide configuration file

The Hands-on Labs deployment process generates the JumpHost configuration file (`config.json`). It contains:

1. Track, Experience and Story metadata, including "resources" and other information copied through from `content.json`
2. A collection of "outputs" from the ARM deployment process. These are keyed by Story ID and output name
3. Other configuration information, including info about O365 tenants, Azure AD user accounts, etc.

Additional Resources

“Hands-on Labs Content Production and Publication” – step by step guide for getting started with the Git version control system.

“Hands-on Labs Content Style Guide” – terminology and presentation guidelines to maintain consistency within the platform.

Appendix A – Creating a Demo

Demos are provisioned at the story level of the content hierarchy and are independent on demand sessions. All Demo stories must be independent of other stories. Adding Demo content to your content package is similar to adding story content. However, there are some important differences as noted below.

experience.json

`experience.json` contains the title, description, permissions, and list of stories which make up an experience. Demos introduce an additional property, "demo_story_ids", to separate the demo content from HOL content. Stories listed under the "story_ids" property will be included in the HOL content package output. Stories listed under the "demo_story_ids" property will be included in the Demo content package output, not HOL. For example:

```
{
    "$schema": "https://immtest-web.azurewebsites.net/schemas/0.0.2/Immersion
ExperienceMetadata.json",
    "title": "Experience No. 1",
    "description": "Lorem ipsum dorem, dolor sit amet",
    "permissions": [ "admins" ] ,
    "story_ids": [
        "story://root/single/experience1/story1",
        "story://root/single/experience1/story2",
        "story://root/single/experience1/demo1/handsonlab"
    ] ,
    "demo_story_ids": [
        "story://root/single/experience1/demo1/demo"
    ]
}
```

content.json

```
{
    "$schema": "https://immtest-web.azurewebsites.net/schemas/0.0.2/Immersion
DemoStoryMetadata.json"
}
```

As for Hands-on Lab stories, `content.json` tracks metadata for a Demo story. Metadata for a demo story includes all of the same properties as an HOL story, plus the following:

Story metadata

1. `target_presenter` – This property sets the visibility of the demo based on the type of logged in user: field, partner, or customer.

Deployment metadata

2. `secondary_templates` – An optional path to additional ARM templates providing resources specific to the requirements of the demo script.

Both of these new properties instruct the platform during Content Processing in how to construct your content package(s).

The following is an example of a single-track content package with two experiences, one with a demo and one without. This demo story has content for both a hands-on lab, and a click-through demonstration. It also has multiple presenter types with content and deployment instructions separated into folders for each type.

```
Single Track Demo Example

root/
    repoconfig.json
    single/
        track.json
        experience1/
            experience.json
            story1/
                content/
                deployment/
                content.json
            demo1/
                content/
                    demo/
                    handsonlab/
                deployment/
                content.json
        experience2/
            experience.json
            demostory/
                content/
                deployment/
                content.json
            holstory/
                content/
                deployment/
                content.json
```

Note: When using the content subfolder to provide both demo and hands-on lab content within the same story, your story URI in the experience metadata should point to the subfolder. For example:

```
{  …
   "story_ids": [
        "story://root/single/experience1/story1",
        "story://root/single/experience1/demo1/handsonlab"
    ] ,
    "demo_story_ids": [
        "story://root/single/experience1/demo1/demo"
    ]
}
```

Demo Deployment

In addition to the information described in the Story Deployment section, the demo content schema allows you to specify additional templates and parameters specific to the target presenter type. For example,

Demo Deployment

```
{
    "permission": [ "everybody" ],
    "primary_template": "template.json",
    "secondary_templates": {
        "field": "field_template.json",
        "partner": "partner_template.json",
        "enduser": "enduser_template.json"
    },
    "parameters": {
        "standalone": "hol.parameters.json",
        "field": "demo.parameters.json",
        "partner": "demo.parameters.json",
        "enduser": "enduser_demo.parameters.json"
    }
}
```

These additional templates allow you to customize the end-state of the lab environment to accommodate the variance in tasks that may be asked of the different presenter types.

Extended Markdown features for Demos

There are additional extensions for Markdown content to specify the separation between the demo "script" and the click-through steps the presenter must follow to highlight the product or feature described. The presenter and click-steps metadata tags must be properly formed XML tags, having property values surrounded by quotes and having an open and closing tag, as shown in the examples below.

Presenter metadata

The `presenter` tag indicates the story text that the presenter will read to describe the feature under demonstration. Both the key and index properties are required.

```
<presenter key="login" index="0">

I'm going to log in to the Azure Portal and then we'll take a look at how easily you
can set up a robust virtual infrastructure in Azure.

</presenter>
```

The `key` attribute is the link between the presenter segment and the click-through steps the demonstrator should follow. These segments will be displayed side-by-side during the demonstration in the Web Guide. The index attribute controls the display order of the segments.

Click-Steps metadata

The `clicksteps` tag surrounds the segments of the Markdown that provide the click-through steps the presenter will follow through the demonstration. The key property is required.

```
<clicksteps key="login">

1. [Open the Azure Portal](https://portal.azure.com)

2. In the **Email or phone** field, enter **<inject key="AzureAdUserEmail"/>**

3. In the **Password** field, enter **<inject key="AzureAdUserPassword"/>**

4. Click **Sign in**

</clicksteps>
```

The `key` attribute must be unique for each pair of segments in the Markdown document. The content processor will identify non-unique values, and report them as errors during processing.

Web Guide Differences

The demo portal offers the end user an in-browser RDP experience and web-based guide for content display. At this time, Launch Helpers are not supported in the web-based guide, and any traditional links included in the Markdown content will be launched in a new tab of the end-user's browser outside the RDP session.

Appendix B – Hands-on Labs Role Based Access

By default, the Hands-on Labs user is given Reader access to the resource group in which the lab is deployed. If additional access is required for your content package, such as virtual machine creation, network modification, or other higher level tasks; we recommend using the resource specific contributor roles. See the below table for the most often used roles.

Role	**Virtual Machine Contributor**
Lets you manage virtual machines, but not access to them, and not the virtual network or storage account they're connected to.	
Role	**SQL Server Contributor**
Lets you manage SQL servers and databases, but not access to them, and not their security -related policies.	
Role	**Storage Account Contributor**
Lets you manage storage accounts, but not access to them.	
Role	**Web Plan Contributor**
Lets you manage the web plans for web sites, but not access to them.	
Role	**Web Site Contributor**
Lets you manage websites (not web plans), but not access to them.	

In addition to the standard RBAC roles available in Azure, we have created the following roles specifically for Hands-on Labs.

Role	**[Hands-on Labs] Data Factory Contributor Additional Permissions**
Additional "write resource group" permission missing from Data Factory Contributor role.	
Actions	
Microsoft.Resources/subscriptions/resourceGroups/write	
NoActions	

Role **[Hands-on Labs] Data Lake Contributor**

Can create and modify Data Lake resources

Actions

Microsoft.DataLakeStore/accounts/*
Microsoft.DataLakeAnalytics/accounts/*

NoActions

Role **[Hands-on Labs] Sql Server Data Masking Contributor**

Can configure data masking rules

Actions

Microsoft.Sql/servers/databases/dataMaskingPolicies/*

NoActions

Role **[Hands-on Labs] Virtual Machine Operator**

Can restart virtual machines

Actions

Microsoft.Compute/virtualMachines/start/action
Microsoft.Compute/virtualMachines/restart/action

NoActions

Appendix C – Recommendations for Self-Paced Labs

If you are building a lab that will primarily target self-paced use, we offer the following recommendations to create the best experience for the end-user.

- Self-paced labs are occasionally also offered by experience on a scheduled basis for large events.
    - If your self-paced lab is not intended to be accompanied by other stories in the track, define your experiences such that you have one story per experience.
    - If your self-paced labs can logically be explored in a group, define your experiences to include all related stories.
- We recommend that you create a single jumphost definition when your stories will operate together. However, we caution against using the same jumphost for more than one track.
- Self-paced labs are executed from a pooled set of provisioned labs.
    - You must request a pool for your lab when you are ready to test in the HOL shared Test environment.
    - If you change the folder structure of your track, you must request new pools for the new structure.
    - Pools are generally kept small in Test and content is refreshed by attrition. As labs are requested, the pool is refilled with the latest available version of the content.
- Self-paced labs are kept in a "warm" state until requested by the end-user. Please let us know if your lab uses any advanced features of Azure that are charged by time rather than use.

Appendix D – Metadata Schema Versions

Version 0.0.1

Name	File Name	Purpose
ImmersionStoryMetadata.json	Content.json	Describe the metadata for a Hands-on Lab story
URI: https://immtest-web.azurewebsites.net/schemas/0.0.1/ImmersionStoryMetadata.json		
ImmersionExperienceMetadata.json	Experience.json	Describe the metadata for a Hands-on Lab experience
URI: https://immtest-web.azurewebsites.net/schemas/0.0.1/ImmersionExperienceMetadata.json		
ImmersionTrackMetadata.json	Track.json	Describe the metadata for a Hands-on Lab track
URI: https://immtest-web.azurewebsites.net/schemas/0.0.1/ImmersionTrackMetadata.json		

Version 0.0.2

Name	File Name	Purpose
ImmersionStoryMetadata.json	Content.json	Describe the metadata for a Hands-on Lab story
URI: https://immtest-web.azurewebsites.net/schemas/0.0.2/ImmersionStoryMetadata.json		
ImmersionDemoStoryMetadata.json	Content.json	Describe the metadata for an Azure Demo story
URI: https://immtest-web.azurewebsites.net/schemas/0.0.2/ImmersionDemoStoryMetadata.json		
ImmersionExperienceMetadata.json	Experience.json	Describe the metadata for either an Azure Demo experience or a Hands-on Lab experience

URI: https://immtest-web.azurewebsites.net/schemas/0.0.2/ImmersionExperienceMetadata.json		
ImmersionTrackMetadata.json	Track.json	Describe the metadata for either an Azure Demo track or a Hands-on Lab track
URI: https://immtest-web.azurewebsites.net/schemas/0.0.2/ImmersionTrackMetadata.json		