

User Guide > Infrastructure Composition Language (ICL)

# Infrastructure Composition Language (ICL)

Spheron Network uses a declarative system for resource allocation. Users specify their deployment requirements, services, datacenters and pricing parameters through a **"manifest"** file called `deploy.yaml`, written in Infrastructure Composition Language (ICL). ICL is designed to be user-friendly and follows the YAML standard, making it similar to Docker Compose files.

The `deploy.yaml` file, which can also use the `.ym1` extension, serves as a request form for network resources.

It's structured into several key sections:

1. [Network Configuration](#)
2. [Version](#)
3. [Services](#)
4. [Profiles](#)
5. [Deployment](#)
6. [GPU Support](#)
7. [Private Container Registry Integration](#)
8. [Port Range](#)
9. [Pull Policy](#)
10. [Shared Memory \(SHM\) Support](#)

**NOTE:** For examples of deployment configurations:

- [Single service deployment](#)
- [Multi-service deployment](#)

These examples demonstrate how to structure your `deploy.yaml` file for different deployment scenarios.

# 1. Network Configuration

The Infrastructure Composition Language (ICL) file allows you to define networking settings for your deployment. This determines how workloads can connect to each other and be accessed externally. By default, workloads within a deployment group are isolated, meaning no external connections are permitted. However, these restrictions can be adjusted as needed.

## 2. Version Configuration

The Spheron configuration file requires a version specification. At present, the only accepted value is "1.0".

## 3. Services Configuration

The `services` section at the top level of the configuration file defines the workloads for your Spheron deployment. It's structured as a map where each key represents a unique service name, and the corresponding value is another map with the following possible fields:

Field Name	Required	Description
image	Yes	Specifies the Docker image for the container. Caution: Using <code>:latest</code> tags is not recommended due to extensive caching by Spheron Providers.
command	No	Defines a custom command to be executed when launching the container.
args	No	Provides arguments for the custom command specified in the 'command' field.
env	No	Sets environment variables for the running container. Refer to the <a href="#">Environment Variables</a> section for more details.

Field Name	Required	Description
expose	No	Determines which entities are permitted to connect to the services. For additional information, see the <a href="#">Port Exposure</a> and <a href="#">Port Range</a> sections.
pull_policy	No	Specifies the image pull policy for the container. For more details, see the <a href="#">Pull Policy</a> section.

## Environment Variables

The `env` field allows you to specify a list of environment variables that will be made available to the running container. These variables are defined in a key-value format. For example:

```
env:  
  - WALLET_ADDR=0xabcddedghijke  
  - VERSION=1.0
```

## Port Exposure

When configuring port exposure for your services, keep these points in mind:

- **HTTPS Support:** Spheron deployments can use HTTPS, but only with self-signed certificates.
- **Signed Certificates:** To implement properly signed certificates, you'll need to use a third-party solution like Cloudflare as a front-end.
- **Flexible Port Mapping:** You're not limited to just port 80 for HTTP/HTTPS ingress. You can expose other ports and map them to port 80 using the `as: 80` directive, provided your application understands HTTP/HTTPS protocols. This is particularly useful for applications like React web apps.
- **Simplified Port Exposure:** In the ICL, it's only required to expose port 80 for web applications. However, this setup specifies that both ports 80 and 443 are exposed.

```
- port: 3000  
  as: 80  
  to:
```

```
- global: true
accept:
- www.yoursite.com
```

## Port Range



**Note:** The port range is only applicable for fizz node deployments when the mode is set to `fizz`. For fizz node deployments, the exposed port cannot be 80 or 443, as fizz nodes don't have an ingress to create subdomain-based deployment links for users.

When deploying to fizz nodes, you can specify a port range using the `port_range` and `port_range_as` fields. Here's an example:

```
- port_range: 8443-8445
  port_range_as: 8443-8445
  to:
    - global: true
```

You can specify either `port`, `port_range`, or both, but make sure to specify at least one of them.



**Important:** For fizz node deployments (mode set to `fizz`), ports 80 and 443 are not available. You must use other port numbers for your services.

The `expose` parameter is a list that defines the connections allowed to the service. Each entry in this list is a map that can include one or more of the following fields:

Field	Required	Description
<code>port</code>	Yes	Specifies the container port that should be made accessible.
<code>as</code>	No	Defines an alternative port number to expose the container port as.
<code>accept</code>	No	Lists the hostnames for which connections should be accepted.

Field	Required	Description
proto	No	Indicates the protocol type. Can be set to either <code>tcp</code> or <code>udp</code> .
to	No	Enumerates the entities permitted to connect to this port. Refer to the <code>expose.to</code> section for more details.

Keep in mind, The `as` parameter determines the default `proto` value.

**NOTE:**

1. If `as` is not specified, it defaults to the value set by the mandatory `port` directive.
2. When `as` is set to 80 (HTTP), the Kubernetes ingress controller automatically makes the application accessible via HTTPS as well. However, this uses the default self-signed ingress certificates.

port	proto default
80	http & https
all others	tcp & udp

## Configuring `expose.to`

The `expose.to` field defines a list of clients allowed to connect to a service. Each item in this list is a map that can contain one or both of these entries:

parameter	Type	Default	Description
service	A service in this deployment	N/A	Permit the specified service to establish a connection.
global	true or false	false	If set to true, allows connections from outside the datacenter

If no service is specified and `global` is set to `true`, any client can connect from any location (this is commonly desired for web servers).

If a service name is specified and `global` is set to `false`, only services within the current datacenter can connect. If a service name is specified and `global` is set to `true`, services from other datacenters within the deployment can connect.

When `global` is set to `false`, a service name must be provided.

## Pull Policy

The `pull_policy` field allows you to specify how the container runtime should handle pulling the image for your service. This can be particularly useful when working with frequently updated images or when you want to ensure you're always using the latest version.

There are three possible values for `pull_policy`:

- `Always`: The image is pulled whenever the pod is started or restarted.
- `IfNotPresent`: The image is pulled only if it's not already present on the node.
- `Never`: The image is never pulled, and the deployment will fail if the image isn't already present on the node.

Example usage:

```
services:
  myapp:
    image: myregistry.com/myuser/myapp:latest
    pull_policy: Always
```

**Note:** If you're using the `:latest` tag for your image, it's recommended to set `pull_policy: Always` to ensure you're always running the most recent version of your image.

## 4. Profiles

The profiles section is used to define named compute and placement profiles that can be referenced in the [deployment section](#). It's also define the name of the deployment, duration of the lease created & the provider tiers. Below is the table describing the parameters:

Field Name	Required	Description
name	No	Specifies the name of the deployment. It's not unique but is useful for users to identify their deployments.
mode	Yes	Defines where you want to deploy your app on. Spheron has 2 mode: <b>provider</b> & <b>fizz</b> . Refer to the <a href="#">Deployment Mode</a> section for more details.
duration	Yes	Defines the duration of the lease. Configured in 1s, 1min, 1h, 1d, 1mon, & 1y. Refer to the <a href="#">Lease Duration</a> section for more details.
tiers	No	Specifies tiers of provider the deployment order need to be matched with. Can be multiple. Refer to the <a href="#">Deployment Tier</a> section for more details.
compute	Yes	Specifies the compute resources that will be allocated to service instances. Refer to the <a href="#">services.env</a> section for more details.
placement	Yes	Outlines the necessary datacenter attributes and the pricing configuration. See the <a href="#">services.expose</a> section for additional information.

## Deployment Mode

Spheron offers two deployment modes:

1. **Provider Mode:** Deploys directly to data center-grade providers. This mode offers:

- Higher stability
- Larger compute resources
- Better bandwidth connections
- Suitable for production-grade applications

- To deploy in this mode, just write `provider` .

2. **Fizz Mode:** Deploys to a network of smaller, consumer-grade devices. This mode offers:

- Lower costs
- Distributed deployment across many nodes
- Less stability compared to Provider Mode
- Suitable for testing or less resource-intensive applications
- To deploy in this mode, just write `fizz` .

Choose the mode that best fits your application's requirements and budget.

## Lease Duration

During the deployment you also pass duration to run the deployment which will be in 1s, 1min, 1h, 1d, 1mon, & 1y. This will specify how much time the lease need to run and accordingly lock funds to continue to run the deployment. When you close the deployment prematurely, then you will unlock the amount that has not been spent.

## Deployment Tier

During deployment, you have the option to specify the tiers on which you want your deployment to be placed, whether on a specific or generalized provider tier. This feature is beneficial for developers who need high reliability and are willing to pay a premium for high-tier providers. Conversely, users with less critical requirements can choose lower-tier providers at a reduced cost. During the testnet phase, there is no premium margin on deployment as we are still finalizing the idea.

We have two general tiers: **Secured** and **Community**.

- **Secured Tier:** This tier consists of high-tier providers who have consistently demonstrated high uptime in the network. It includes Provider Tiers 1 to 3.
- **Community Tier:** This tier consists of lower-tier providers who have recently joined the network or have less reliable hardware. It includes Provider Tiers 4 to 7.

To deploy your services on any tier, use the following values:



Tier	Details
secured	Can be deployed on Provider Tiers 1 to 3.
community	Can be deployed on Provider Tiers 4 to 7.
secured-1	Specifically deployed on Provider Tier 1.
secured-2	Specifically deployed on Provider Tier 2.
secured-3	Specifically deployed on Provider Tier 3.
community-1	Specifically deployed on Provider Tier 4.
community-2	Specifically deployed on Provider Tier 5.
community-3	Specifically deployed on Provider Tier 6.
community-4	Specifically deployed on Provider Tier 7.

**Note:** Users can specify multiple tiers during deployment, and the matchmaker will select the best possible provider based on the specified requirements and other parameters.

## Compute Profiles

Within `profiles.compute`, you can create a map of named compute profiles. Each profile specifies the compute resources that will be allocated to service instances using that profile.

### Example

This defines a profile named `api` with resource requirements of 2 vCPUs, 4 gigabytes of memory, and 20 gigabytes of storage.

```
api:  
  cpu: 2  
  memory: "4Gi"  
  storage: "20Gi"
```

`cpu` units indicate a vCPU share, which can be fractional. Without a suffix, the value denotes a fraction of a whole CPU share. If an `m` suffix is used, the value specifies the number of milli-CPU shares, which equals 1/1000 of a CPU share.

**Example**

Value	CPU-Share
1	1
0.5	1/2
"100m"	1/10
"50m"	1/20

`memory` and `storage` units are expressed in terms of bytes, with specific suffixes utilized to simplify their representation as follows:

Suffix	Value
k	1000
Ki	1024
M	1000^2
Mi	1024^2
G	1000^3
Gi	1024^3
T	1000^4
Ti	1024^4
P	1000^5
Pi	1024^5
E	1000^6
Ei	1024^6

## Placement Profiles

`profiles.placement` is a map of named datacenter profiles. Each profile outlines the necessary datacenter attributes and the pricing configuration for each compute profile that will be utilized within the datacenter.

**Note:** For Spheron CLI versions 1.2.0 and later:

- The `precision` is no longer required based on the amount value.
- The `denom` field has been replaced with `token`.

If you're using an older version, please upgrade to the latest version using the instructions in our [CLI upgrade guide](#).

### Example

```
useast:
  attributes:
    region: us-east
  pricing:
    web:
      token: USDT
      amount: 80
```

This defines a profile named `useast` with the required attributes `{region="us-east"}` and a maximum price of 80 USDT per hour for the web compute profiles.

Providers can only assign themselves to a predefined set of regions. If you specify an unlisted region, no provider will bid on your deployment, causing it to fail automatically. Please refer to the list mentioned in the [Supported Regions](#) section.

**i** To learn more about advanced placement attributes and fine-grained control over your deployments, check out the [Advanced Placement Attributes](#) section at the end of this guide.

## 5. Deployment Configuration

The `deployment` section is where you define the specific deployment strategy for your services. It's structured as a map, with each service name corresponding to its deployment configuration.

For each service you want to deploy, you create an entry in the `deployment` section. This entry combines datacenter profiles with compute profiles to specify the final resource configuration for the service.

### Example

```
api:
  useast:
    profile: api
    count: 30
```

This specifies that 30 instances of the `api` service should be deployed to a datacenter that matches the `useast` datacenter profile. Each instance will have access to the resources defined in the `api` compute profile.

## 6. GPU Integration Support in Compute Profiles

You can add GPUs to your workload by including them in the compute profile section. The placement of the GPU stanza is illustrated in the full compute profile example below.

**Note:** When specifying the GPU model, such as `h100` in this example, ensure that the model name matches the conventions listed in the provided reference.

```
profiles:
  compute:
    gpudep:
      resources:
        cpu:
          units: 8.0
        memory:
          size: 12Gi
        storage:
          size: 10Gi
        gpu:
          units: 1
```

```
attributes:
  vendor:
    nvidia:
      - model: a100
```

## Additional GPU Use Notes

### Complete GPU ICL Example

For a comprehensive example of a GPU-enabled ICL, refer to this [example](#) which includes the declaration of several GPU models.

### Optional Model Specification

Specifying a GPU model in the ICL is optional. If your deployment does not require a specific GPU model, you can leave the model declaration blank, as demonstrated in the following example.

```
gpu:
  units: 1
  attributes:
    vendor:
      nvidia:
```

### Declaring Multiple Models

If your deployment is optimized to run on multiple GPU models, include the relevant list of models as shown in the example below. In this setup, any Spheron provider with a listed model will bid on the deployment.

```
gpu:
  units: 1
  attributes:
    vendor:
      nvidia:
        - model: rtx4090
        - model: t4
```

## Specifying GPU RAM

Optionally, the ICL can include a GPU RAM/VRAM requirement, as illustrated in the example below.

```
gpu:
  units: 1
  attributes:
    vendor:
      nvidia:
        - model: a100
          ram: 80Gi
```

## Specifying GPU Interface

Optionally, the ICL can include a GPU interface requirement, as shown in the example below.

**Note:** Only the values `pcie` or `sxm` should be used in the Spheron ICL. There are several variants of the SXM interface, but only the simple `sxm` value should be used in the ICL.

```
gpu:
  units: 1
  attributes:
    vendor:
      nvidia:
        - model: h100
          interface: sxm
```

## Specifying GPU with RAM and Interface

Here is an example of specifying both RAM and interface in the ICL GPU section.

```
gpu:
  units: 1
  attributes:
    vendor:
      nvidia:
        - model: h100
```

```
interface: pcie
ram: 90Gi
```

**Note:** For detailed information on GPU support and the corresponding model names, please refer to the [GPU support page](#).

## 7. Private Container Registry Integration

Spheron Network now supports private container registries, allowing you to use images from your private repositories securely in your deployments. This feature enhances security and flexibility for users who need to work with proprietary or sensitive container images.

### Configuring Private Registry Access

To use images from a private registry, you'll need to provide authentication details in your `deploy.yaml` file. Here's an example of how to structure this information:

```
services:
  myapp:
    image: myregistry.com/myuser/myapp:latest
    credentials:
      registry: myregistry.com
      username: myusername
      password: "mysecretpassword"
    expose:
      - port: 3000
        as: 80
        to:
          - global: true
```

### Important Notes:

#### 1. Registry Specification:

- For Docker Hub, use `docker.io`
- For GitHub Container Registry, use `ghcr.io`
- For Gitlab Container Registry, use `registry.gitlab.com`

- For AWS ECR, use `public.ecr.aws`
- For Azure Container Registry, use `myregistry.azurecr.io`

## 2. Authentication:


- Docker Hub: Use your account password in the `password` field
- GitHub Container Registry: Use a Personal Access Token with appropriate permissions in the `password` field

**3. Compatibility:** This feature has been tested with Docker Hub and GitHub Container Registry. Other registries may work but are not officially supported.

By leveraging private registry support, you can maintain tighter control over your container images while benefiting from Spheron's robust deployment infrastructure.

 Remember to keep your authentication credentials secure and never commit them directly to your version control system.

# 8. Port Range

 **Note:** Port range only applies to fizz node deployments when the mode is set to `fizz`. The exposed port cannot be 80 or 443 for fizz node deployments, as fizz nodes don't have an ingress to create subdomain-based deployment links for users.

When deploying to fizz nodes, you can specify a port range using the `port_range` and `port_range_as` fields. This feature allows you to expose multiple consecutive ports for your service. Here's an example:

```
expose:
- port_range: 8443-8445
  port_range_as: 8443-8445
  to:
    - global: true
```

Key points about using port range:



1. You can specify either `port` or `port_range` or both, but make sure to specify at least one of them.
2. The `port_range` field defines the range of ports to be exposed from the container.
3. The `port_range_as` field defines how these ports should be mapped externally.
4. The range is inclusive, so ports 8443, 8444, and 8445 would be exposed in the example above.

**i Important:** For fizz node deployments (mode set to `fizz`), ports 80 and 443 are not available. You must use other port numbers for your services.

## 9. Pull Policy

The `pull_policy` field allows you to specify how the container runtime should handle pulling the image for your service. This can be particularly useful when working with frequently updated images or when you want to ensure you're always using the latest version.

There are three possible values for `pull_policy` :

- `Always` : The image is pulled every time the pod is started or restarted.
- `IfNotPresent` : The image is pulled only if it's not already present on the node.
- `Never` : The image is never pulled, and the deployment will fail if the image isn't already present on the node.

Example usage:

```
services:
  myapp:
    image: myregistry.com/myuser/myapp:latest
    pull_policy: Always
```

**i Note:** If you're using the `:latest` tag for your image, it's recommended to set `pull_policy: Always` to ensure you're always running the most recent version of your image.

Using the pull policy feature give you more control over how and when your container images are updated, which can be crucial for maintaining consistency across deployments or ensuring you're always running the latest version of your application.

## 10. Shared Memory (SHM) Support

Spheron's Infrastructure Composition Language (ICL) now supports the configuration of Shared Memory (SHM) for services that require inter-process communication or temporary file storage with high-speed access. This feature is particularly useful for applications that need to share data quickly between multiple processes within the same container.

### Configuring SHM in Compute Profiles

To enable SHM, you can add a new storage class named `ram` in your compute profile definition. Here's an example of how to include SHM in your ICL:

```
profiles:
  compute:
    myapp:
      resources:
        cpu:
          units: 2
        memory:
          size: 2Gi
        storage:
          - size: 10Gi
          - name: sharedmem
            size: 2Gi
            attributes:
              persistent: false
              class: ram
```

In this example, we've defined a compute profile named `datavis` that includes:

- Standard storage of 10Gi
- A shared memory storage named `sharedmem` of 2Gi using the `ram` class

### Important Notes:

1. SHM must be non-persistent. The ICL validation will raise an error if SHM is defined as persistent.
2. The `class: ram` attribute is used to specify that this storage should be treated as shared memory.
3. The name of the shared memory storage (e.g., `sharedmem`) can only contain alphanumeric characters and hyphens (-). Underscores (\_) are not allowed.

## Using SHM in Services

Once you've defined the SHM in your compute profile, you can use it in your service definition:

```
services:
  visualization:
    image: myregistry.com/datavis:latest
    expose:
      - port: 8080
        as: 80
        to:
          - global: true
    params:
      storage:
        sharedmem:
          mount: /dev/shm
```

In this service definition:

- We're using the `visualization` service with a specified Docker image.
- The `params.storage.shared_mem.mount` field is used to mount the shared memory to `/dev/shm` in the container.

## Benefits of Using SHM

1. **High-speed Inter-process Communication:** SHM allows rapid data sharing between processes in the same container.
2. **Temporary File Storage:** It provides a fast storage option for temporary files that don't need to persist beyond the container's lifecycle.
3. **Resource Efficiency:** By using memory for storage, you can reduce I/O operations and improve overall application performance.

**Note:** When using SHM, be mindful of your memory usage. Excessive use of shared memory can impact the overall performance of your application and other services running on the same node.

By leveraging Shared Memory in your Spheron deployments, you can optimize performance for applications that require fast, temporary storage or efficient inter-process communication.

## 11. Advanced Placement Attributes

Spheron allows you to precisely control where your applications are deployed using advanced placement attributes. These settings help you target specific regions, providers, or nodes for your deployments.

### Available Attributes

#### 1. Region Selection

Use the `region` attribute to specify which geographic region your application should be deployed in:

```
attributes:  
  region: us-east
```

This is particularly useful when you need to:

- Minimize latency for users in specific geographic areas
- Ensure your application runs in a specific region

**i** The region must match one of Spheron's [supported region codes](#). If not specified, your application can be deployed in any available region.


#### 2. Provider Selection

The `desired_provider` attribute lets you specify a particular provider using their blockchain address:

```
attributes:  
  desired_provider: "0x1234...5678" # Replace with actual provider address
```

This is helpful when you:

- Have had good experiences with a specific provider
- Want to maintain consistency across deployments
- Need specific hardware or capabilities that a provider offers

 This works in both `provider` and `fizz` modes. In `fizz` mode, it will select `fizz` nodes that are connected to your chosen provider.

### 3. Fizz Node Selection

For deployments using `fizz` mode, you can target a specific `fizz` node:

```
attributes:  
  desired_fizz: "0xabcd...ef12" # Replace with actual fizz node address
```

This is useful when you:

- Want to deploy to a specific node you trust
- Need to maintain application state on a particular node
- Are testing node-specific functionality

## Combining Attributes

You can use multiple attributes together for precise control. Here's a comprehensive example:

```
profiles:  
  placement:  
    useast:
```

**attributes:****region:** `us-east`

# Geographic region

**desired\_provider:** `"0x1234...5678"`

# Specific provider

**desired\_fizz:** `"0xabcd...ef12"`

# Specific fizz node

**pricing:****web:****token:** `USDT`**amount:** `8`**Important Considerations:**

- When using `desired_fizz`, make sure your deployment mode is set to `fizz`
- If combining `desired_provider` and `desired_fizz`, verify that the fizz node is connected to the specified provider
- Invalid combinations will cause your deployment to fail

These placement attributes give you granular control over your deployments while maintaining the flexibility to scale across Spheron's network when needed.

---

[< Deploy Your App](#)[Protocol CLI >](#)

Last updated on November 27, 2024