# What is SpatialOS?

SpatialOS is a distributed, cloud-based platform for game development and operation that integrates with and augments existing workflows, engines and tools.

SpatialOS empowers developers to build, run and operate previously impossible online games. As long as you can express the world in terms of independent entities that have a location in space, you can use SpatialOS to simulate it.

## How it works

### Creating the world

You set up your SpatialOS world by specifying what the objects in it are. These will be the world’s entities: all meaningful objects in the world are entities.

Each entity consists of one or more components; each component adds additional behaviour or functionality. This is similar to the [entity-component-system](https://en.wikipedia.org/wiki/Entity%E2%80%93component%E2%80%93system) architectural pattern, common in game development. Instead of systems, SpatialOS has workers, in an entity-component-worker architecture. Workers are described in more detail below.

To start with, you create a snapshot that lists all of the entities in the world and what their component properties are (for example, their position in the world). Once you’ve done that, you can start the world running in a deployment.

### 

### Once the world is running

In a running deployment, SpatialOS manages and keeps track of the world, but it doesn’t actively make changes to entities and their components. Only **workers** can make changes to the world. Workers are programs that you create, that read from and write to the entities in the world.

Because a worker is just a program, it can do a lot of things that aren’t related to the SpatialOS world - like executing game logic, or communicating with external databases. If it wants to interact with the SpatialOS world, it must use the SpatialOS APIs to do that.

## Managed vs external workers

### Workers can be managed or external.

**Managed workers** are called managed workers because their lifecycle is managed by SpatialOS. Based on how many instances of the worker the world needs, SpatialOS runs worker instances, and allocates them to different parts of the world.

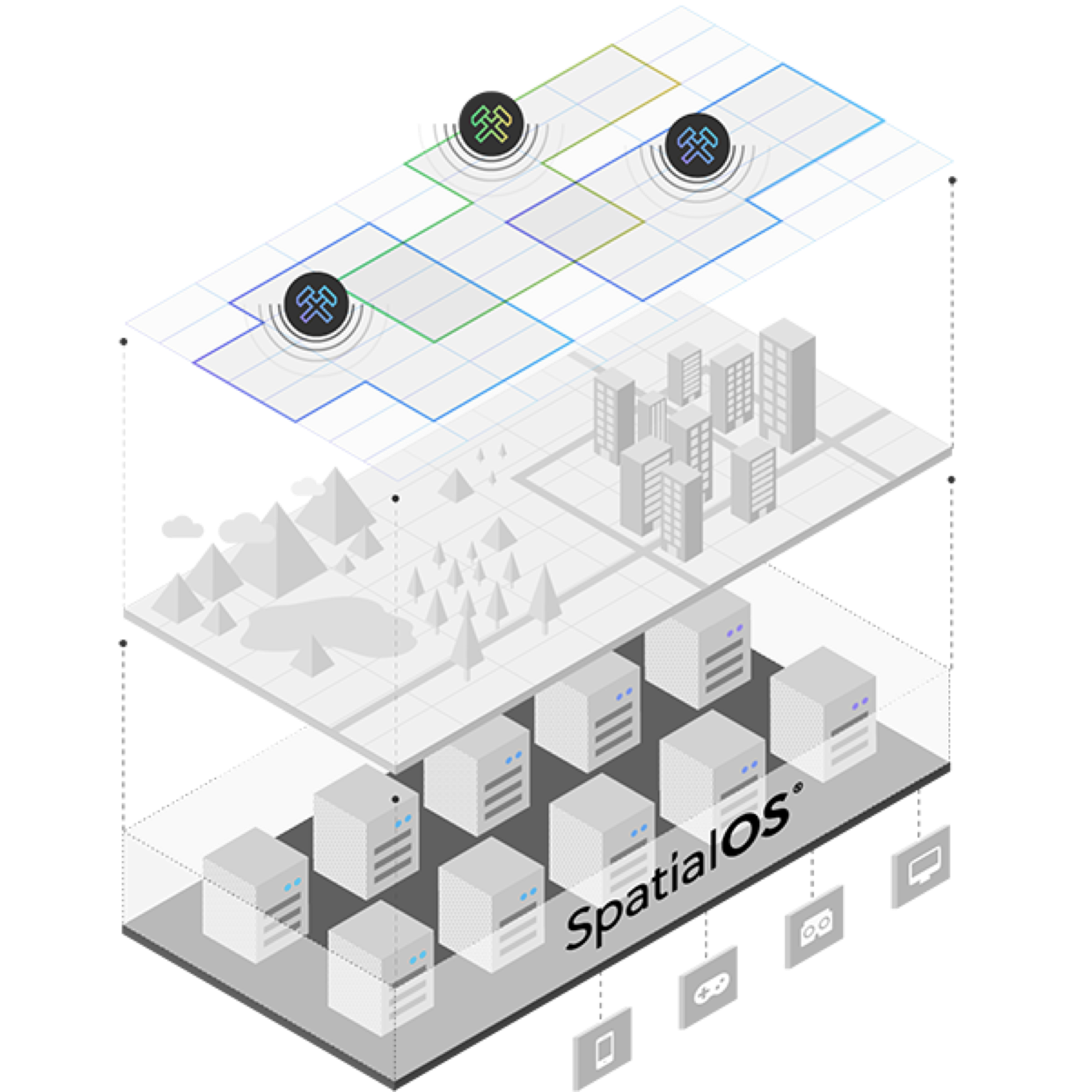
If a worker instance gets overloaded, SpatialOS can start up another worker instance to help; if there’s too little work to do, it can shut down worker instances. This is called load-balancing, and it’s one of the things that means SpatialOS can run massive virtual worlds. You can control some aspects of load-balancing, so it works in the right way for your world.

**External workers** are usually game clients. They aren’t managed by SpatialOS: instead, their lifecycle is controlled by a player connecting or disconnecting.

### How workers interact with the world

It’s important to understand that workers only know about a part of the world. Managed workers know about the part of the world that SpatialOS allocates to them, plus a region around those areas. External workers are often configured to know about the part of the world around the player’s entity.

This means that, when you’re writing code for workers, you must write as though you only know about the world around you, for any arbitrary section of the world. SpatialOS automatically sends workers information about what’s happening to the entities within the areas they know about. If you want information about the world further away, the worker must query SpatialOS for it.



### 

### Overlap

In the diagram, you can see that more than one worker instance knows about the same part of the world. In order to prevent two workers trying to make changes at the same time, only one worker instance at a time is allowed to write to a given component of a given entity. Many workers may be able to read from the entity; but only one can write to that particular component on that particular entity.

Entities can freely move into and out of an area that a worker instance knows about. SpatialOS manages this handover of entity data seamlessly, so players shouldn’t notice when this is happening.

### The state of the world

It’s important to understand that canonical state of the world is in SpatialOS, not in the worker. This matters because it means that workers need to send information to SpatialOS. You can keep information locally on a worker, if you only need that worker to know about it.

But if any other worker will ever need to know that information, it must be sent to SpatialOS.

**Developing with SpatialOS**

There are SDKs that let you write workers in several different languages: C++, C#, and Java.

There are also SDKs so you can write workers using the game engines [Unity3D](https://unity3d.com/) and [Unreal Engine](https://www.unrealengine.com). It’s also possible to use the other SDKs to build your own integration with other game engines.

When you’re developing a worker, you can write the worker code and try it out in a local deployment for debugging and testing. This is very similar to running a SpatialOS deployment in the cloud - just smaller. When you’re ready to try running your world at scale, you can deploy to the cloud, which also means you can make it available to external players.