Cheat Sheet: Django Application Development with SQL and Databases

Estimated reading time: 12 minutes

Package/Method	Description	Code Example
all()	Retrieves all instances of the 'MyModel' model from the database.	`MyModel.objects.all()` Copied!
AVG	Calculates the average value of a column.	`SELECT AVG(column1) FROM table_name;` Copied!
Avg()	Calculates the average of a field.	`MyModel.objects.aggregate(Avg('field'))` Copied!

Basic View Function	Function-based view that returns "Hello, World!" From Django.http import HttpResponse	`def my_view(request):`
		`# Your view logic here`
		`return HttpResponse("Hello, World!")`
		Copied!

Bootstrap classes and components	Create visually appealing and responsive web pages without having to write CSS styles manually.	`<a href="#" class="btn btn-primary">Click Me</a>` Copied!
Bootstrap CSS	Link to include Bootstrap CSS in the base template.	Add the following link to the `<head>` section of your base template (usually base.html):

Package/Method	Description	Code Example
		`<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css" rel="stylesheet">` Copied!

Bootstrap JavaScript	Script tag to include Bootstrap JavaScript library.	`Include the Bootstrap JavaScript library at the end of the <body> section to enable certain features (for example, dropdowns, modals):`
		`<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/js/bootstrap.min.js"></script>`
		Copied!

Collecting static files	When deploying your project, you need to collect all static files into a single location.	`python manage.py collectstatic`
		`STATIC_ROOT = os.path.join(BASE_DIR, 'staticfiles')`
		Copied!

Configuration – App Dirs	A configuration option used within the TEMPLATES setting. When set to TRUE, Django will look for template files within the app directories.	`Make sure the APP_DIRS setting is set to True in the TEMPLATES list. This allows Django to look for static files within the apps' directories.`

Package/Method	Description	Code Example

		`TEMPLATES = [`
		`{`
		`# ...`
		`APP_DIRS': True,`
		`# ...`
		`},`
		`]`
		Copied!

Configuration – Installed apps	Defines a list of all the applications installed in the project.	Add 'django.contrib.staticfiles' to your INSTALLED_APPS in settings.py:
		`INSTALLED_APPS = [`
		`# ...`
		`django.contrib.staticfiles',`
		`# ...`
		`]`
		Copied!

Configuration – Static files	Django settings for static files configuration.	In your Django settings (settings.py), define the following settings:

Package/Method	Description	Code Example

		`STATIC_URL = 'https://prod-edx-edxapp-assets.edx-cdn.org/static/studio/edx.org-next/' # URL to access static files'`
		`STATICFILES_DIRS = [os.path.join(BASE_DIR, 'static')] # Directory to look for static files`
		Copied!

contains	Checks if the value is a substring within the field.	`MyModel.objects.filter(field__contains="value")` Copied!
COUNT	Counts the number of rows or non-null values in a column.	`SELECT COUNT(*) FROM table_name; or SELECT COUNT(column1) FROM table_name;` Copied!
count()	Counts the number of objects.	`MyModel.objects.count()` Copied!

CreateView	Displays a form to create a new object.	`class MyCreateView(CreateView):`
		`model = MyModel`
		`template_name = 'my_template.html'`
		`fields = '__all__' # or specify a list of fields`
		Copied!

DELETE FROM	Deletes data from a table based on specified conditions.	`DELETE FROM table_name WHERE condition;` Copied!
delete()	Deletes an object.	

Package/Method	Description	Code Example
		`obj.delete()` Copied!

DeleteView	Displays a confirmation page to delete an object.	`class MyDeleteView(DeleteView):`
		`model = MyModel`
		`template_name = 'my_template.html'`
		`success_url = '/success-url/'`
		`pk_url_kwarg = 'my_model_id' # default: pk`
		Copied!

DetailView	Displays details of a single object.	`class MyDetailView(DetailView):`
		`model = MyModel`
		`template_name = 'my_template.html'`
		`context_object_name = 'object' # default: object`
		`pk_url_kwarg = 'my_model_id' # default: pk`
		Copied!

DISTINCT	Returns unique values from a column.	`SELECT DISTINCT column1 FROM table_name;` Copied!
django.db.models.Model	Define a model.	

Package/Method	Description	Code Example

		`from django.db import models`
		`class MyModel(models.Model):`
		`field1 = models.CharField(max_length=100)`
		`field2 = models.IntegerField()`
		Copied!

endswith	Determines whether a string ends with the specified suffix.	`MyModel.objects.filter(field__endswith="value")` Copied!
exact	Retrieves instances of the 'MyModel' model from the database where the value of the 'field' attribute is exactly equal to "value".	`MyModel.objects.filter(field__exact="value")` Copied!
field	Performs a filtering operation on the 'MyModel' model instances based on a related model's field value.	`MyModel.objects.filter(related_model__field="value")` Copied!

filter()	Filter objects using conditions.	`MyModel.objects.filter(field1="value")`
		`MyModel.objects.filter(field2__gt=5)`
		Copied!

filter(ForeignKey)	Performs conditional joins.	`MyModel.objects.filter(related_model__isnull=True)` Copied!
FROM	Specifies the table from which data is retrieved.	`SELECT column1, column2 FROM table_name;`

Package/Method	Description	Code Example
		Copied!
FULL JOIN	Returns all rows from both tables, regardless of the match.	`SELECT column1, column2 FROM table1 FULL JOIN table2 ON table1.column = table2.column;` Copied!
get()	Retrieves a single instance of the 'MyModel' model from the database where the value of 'field1' is "value".	`MyModel.objects.get(field1="value")` Copied!
GROUP BY	Groups rows based on a specified column.	`SELECT column1, COUNT(*) FROM table_name GROUP BY column1;` Copied!
gt	Checks if the value of 'field' is numerically greater than 5.	`MyModel.objects.filter(field__gt=5)` Copied!
Handle a Form Submission	Function-based view to handle form submission. From django.shortcuts import render	`def my_form_view(request):` `if request.method == 'POST':` `# Process the form data here` `else:` `# Display the form` `return render(request, 'my_form_template.html', context)`

Package/Method	Description	Code Example
		Copied!
Handle URL Parameters	Function-based view that accesses URL parameters.	`def my_param_view(request, param):` `# Access the 'param' value from the URL` Copied!
HAVING	Filters grouped data based on specified conditions.	`SELECT column1, COUNT(*) FROM table_name GROUP BY column1 HAVING COUNT(*) > 1;` Copied!
iexact	The iexact lookup is case-insensitive, meaning it will match values regardless of whether they are uppercase or lowercase and provide a case-insensitive match.	`MyModel.objects.filter(field__iexact="value")` Copied!
in	Checks if the value of the field is present in the given list of values.	`MyModel.objects.filter(field__in=["value1", "value2"])` Copied!
INNER JOIN	Returns only matching rows from both tables.	`SELECT column1, column2 FROM table1 INNER JOIN table2 ON table1.column = table2.column;` Copied!
INSERT INTO	Inserts data into a table.	`INSERT INTO table_name (column1, column2) VALUES (value1,` `value2);`

Package/Method	Description	Code Example
		Copied!

JOIN	Combines rows from multiple tables based on related columns.	`SELECT column1, column2 FROM table1 JOIN table2 ON`
		`table1.column = table2.column;`
		Copied!

LEFT JOIN	Returns all rows from the left table and matching rows from the right table.	`SELECT column1, column2 FROM table1 LEFT JOIN table2 ON`
		`table1.column = table2.column;`
		Copied!

ListView:	Displays a list of objects.	`class MyListView(ListView):`
		`model = MyModel`
		`template_name = 'my_template.html'`
		`context_object_name = 'object_list' # default:`
		`object_list`
		Copied!

lt	Checks if the value of ‘field’ is numerically less than 10.	`MyModel.objects.filter(field__lt=10)` Copied!
makemigrations /migrate	Create database tables based on models.	

Package/Method	Description	Code Example

		`python manage.py makemigrations`
		
		`python manage.py migrate`
		Copied!

many_to_many	Performs many-to-many join.	`obj.many_to_many_field.all()` Copied!
MAX	Finds the maximum value in a column.	`SELECT MAX(column1) FROM table_name;` Copied!
Max()	Provides the maximum value of a field.	`MyModel.objects.aggregate(Max('field'))` Copied!
MIN	Finds the minimum value in a column.	`SELECT MIN(column1) FROM table_name;` Copied!
Min()	Provides the minimum value of a field.	`MyModel.objects.aggregate(Min('field'))` Copied!

obj = MyModel(field1="value", field2=5)
 	
obj.save()
	Creates a new instance of the 'MyModel' model with the values "value" for 'field1' and 5 for 'field2', and then saves the instance to the database.	`obj = MyModel(field1="value", field2=5)`
		`obj.save()`
		Copied!

obj.field1 = "new value"
obj.save()
	Updates the value of 'field1' for the 'obj' instance to "new value" and saves the changes to the database.	`obj.field1 = "new value"`

Package/Method	Description	Code Example
		`obj.save()` Copied!
obj.model_set.all()	Fetches all related objects associated with the 'obj' instance. Access related objects in reverse (ForeignKey)	`obj.model_set.all()` Copied!
obj.related_model	Retrieves the related model associated with the 'obj' instance. Access related objects (Foreign Key or OneToOneField)	`obj.related_model` Copied!
ORDER BY	Sorts the result set based on specified columns in ascending or descending order.	`SELECT column1, column2 FROM table_name ORDER BY column1 ASC;` Copied!
order_by()	Orders objects based on a field.	`MyModel.objects.order_by('field')` Copied!
order_by(-)	Order objects based on fields in descending order.	`MyModel.objects.order_by('-field')` Copied!
prefetch_related	Performs left Outer join.	`MyModel.objects.prefetch_related('related_model')` Copied!

Protecting Views (Restrict Access) using @login_required Decorator	Function-based view protected with login_required decorator. From django.contrib.auth.decorators import login_required	`@login_required`
		`def my_protected_view(request):`

Package/Method	Description	Code Example
		`# Your view logic here` Copied!

Redirect to a URL	Function-based view to redirect to a specific URL.
	From django.shortcuts import redirect
		`def my_redirect_view(request):`
		`return redirect('url_name_or_path')`
		Copied!

Render a Template	Function-based view to render a template with context.
	From django.shortcuts import render
		`def my_template_view(request):`
		`context = {'variable': value}`
		`return render(request, 'my_template.html', context)`
		Copied!

RIGHT JOIN	Returns all rows from the right table and matching rows from the left table.	`SELECT column1, column2 FROM table1 RIGHT JOIN table2 ON table1.column = table2.column;` Copied!
SELECT	Retrieves data from one or more tables based on specified columns.	`SELECT column1, column2 FROM table_name;` Copied!
select_related	Performs inner join.	`MyModel.objects.select_related('related_model')` Copied!
startswith	Determines whether a string begins with the characters of a specified string.	`MyModel.objects.filter(field__startswith="value")` Copied!
SUM	Calculates the sum of values in a column.	`SELECT SUM(column1) FROM table_name;`

Package/Method	Description	Code Example
		Copied!
Sum()	Provides the sum of a field.	`MyModel.objects.aggregate(Sum('field'))` Copied!
UPDATE	Modifies data in a table based on specified conditions.	`UPDATE table_name SET column1 = value1 WHERE condition;` Copied!

UpdateView	Displays a form to update an existing object.	`class MyUpdateView(UpdateView):`
		`model = MyModel`
		`template_name = 'my_template.html'`
		`fields = '__all__' # or specify a list of fields`
		`pk_url_kwarg = 'my_model_id' # default: pk`
		Copied!

Usage – Static content	Code to style the HTML templates and provide interactivity to web pages.	`<link href="{% static 'your_app/css/style.css' %}" rel="stylesheet">`
		`<script src="{% static 'your_app/js/script.js' %}"></script>`
		`<img src="{% static 'your_app/img/logo.png' %}" alt="Logo">`
		Copied!

Package/Method	Description	Code Example
WHERE	Filters data based on specified conditions.	`SELECT column1, column2 FROM table_name WHERE condition;`

Glossary: Introduction to Containers w/ Docker, Kubernetes & OpenShift

Welcome! This alphabetized glossary contains many of the terms in this course. This comprehensive glossary also includes additional industry-recognized terms not used in course videos. These terms are essential for you to recognize when working in the industry, participating in user groups, and in other professional certificate programs.

Estimated reading time: 20 minutes

Term	Definition
A container	powered by the containerization engine, is a standard unit of software that encapsulates the application code, runtime, system tools, system libraries, and settings necessary for programmers to efficiently build, ship and run applications.
A Dockerfile	is a text document that contains all the commands you would normally execute manually in order to build a Docker image. Docker can build images automatically by reading the instructions from a Dockerfile.
A/B testing	Strategy is mostly used for testing new features in front-end applications. It is used to evaluate two versions of the application namely A and B, to assess which one performs better in a controlled environment. The two versions of the applications differ in terms of features and cater to different sets of users. Based on the interaction and responses received from the users such as feedback, you can choose one of the versions of the application that can be deployed globally into production.
Agile	is an iterative approach to project management and software development that helps teams deliver value to their customers faster and with fewer issues.
Automated bin packing	Increases resource utilization and cost savings using a mix of critical and best-effort workloads.
Batch execution	Manages batch and continuous integration workloads and automatically replaces failed containers, if configured.

Term	Definition
Build	The process of transforming inputs into a resultant object.
BuildConfig	An OpenShift-specific object that defines the process for a build to follow. The build process makes use of the input sources and the build strategy. The BuildConfig is the blueprint, and the build is an instance of that blueprint.
Canary Deployments	Aims to deploy the new version of the application by gradually increasing the number of users. The canary deployment strategy uses the real users to test the new version of the application. As a result, bugs and issues can be detected and fixed before the new version of the application is deployed globally for all the users.
CI/CD pipelines	A continuous integration and continuous deployment (CI/CD) pipeline is a series of steps that must be performed in order to deliver a new version of software. CI/CD pipelines are a practice focused on improving software delivery throughout the software development life cycle via automation.
Circuit breaking	A method to prevent errors in one microservice from cascading to other microservices.
Client-server architecture	is a distributed application structure that partitions tasks or workloads between the providers of a resource or service, called servers, and service requesters, called clients.
Cloud Controller Manager	A Kubernetes control plane component that embeds cloud-specific control logic. The cloud controller manager lets you link your cluster into your cloud provider's API, and separates out the components that interact with that cloud platform from components that only interact with your cluster.
Cloud native	A cloud-native application is a program that is designed for a cloud computing architecture. These applications are run and hosted in the cloud and are designed to capitalize on the inherent characteristics of a cloud computing software delivery model.
Cluster Autoscaler	Also known as CA. An API resource that autoscales the cluster itself, increasing and decreasing the number of available nodes that pods can run on.
Cluster	A set of worker machines, called nodes, that run containerized applications. Every cluster has at least one worker node.
Config Map	An API object used to store non-confidential data in key-value pairs. Pods can consume ConfigMaps as environment variables, command-line arguments, or as configuration files in a volume.
Configuration Change	A trigger that causes a new build to run when a new BuildConfig resource is created.

Term	Definition
Container Orchestration	Container orchestration is a process that automates the container lifecycle of containerized applications.
Container Registry	Used for the storage and distribution of named container images. While many features can be built on top of a registry, its most basic functions are to store images and retrieve them.
Container Runtime	The container runtime is the software that is responsible for running containers.
Control Loop	A non-terminating loop that regulates the state of a system. A thermostat is an example of a control loop.
Control plane	The container orchestration layer that exposes the API and interfaces to define, deploy, and manage the lifecycle of containers.
Controller	In Kubernetes, controllers are control loops that watch the state of your cluster, then make or request changes where needed. Each controller tries to move the current cluster state closer to the desired state.
CRDs	Custom code that defines a resource to add to your Kubernetes API server without building a complete custom server.
Custom build strategy	Requires you to define and create your own builder image.
Custom builder images	Are regular Docker images that contain the logic needed to transform the inputs into the expected output.
Custom controllers	Reconcile the custom resources (CRDs) actual state with its desired state.
Daemon-less	A container runtime that does not run any specific program (daemon) to create objects, such as images, containers, networks, and volumes.
DaemonSet	Ensures a copy of a Pod is running across a set of nodes in a cluster.
Data (Worker) Plane	The layer that provides capacity such as CPU, memory, network, and storage so that the containers can run and connect to a network.
Data plane	Communication between services is handled by the data plane. If a service mesh is absent, the network cannot identify the type of traffic that flows, the source, and the destination and make any necessary decisions.
Declarative Management	A desired state that can be expressed (for example, the number of replicas of a specific application),and Kubernetes will actively work to ensure that the observed state matches the desired state.

Term	Definition
Deployment	An object that provides updates for both Pods and ReplicaSets. Deployments run multiple replicas of an application by creating ReplicaSets and offering additional management capabilities on top of those ReplicaSets. In addition, deployments are suitable for stateless applications.
Designed for extensibility	Adds features to your cluster without adding or modifying source code.
DevOps	is a set of practices, tools, and a cultural philosophy that automate and integrate the processes between software development and IT teams.
Docker client	is the primary way that many Docker users interact with Docker. When you use commands such as docker run, the client sends these commands to dockerd, which carries them out. The docker command uses the Docker API. The Docker client can communicate with more than one daemon.
Docker Command Line Interface (CLI)	The Docker client provides a command line interface (CLI) that allows you to issue build, run, and stop application commands to a Docker daemon.
Docker daemon (dockerd)	creates and manages Docker objects, such as images, containers, networks, and volumes.
Docker Hub	is the world's easiest way to create, manage, and deliver your team's container applications.
Docker localhost	Docker provides a host network which lets containers share your host's networking stack. This approach means that a localhost in a container resolves to the physical host, instead of the container itself.
Docker networks	help isolate container communications.
Docker plugins	such as a storage plugin, provides the ability to connect external storage platforms.
Docker remote host	A remote Docker host is a machine, inside or outside our local network which is running a Docker Engine and has ports exposed for querying the Engine API.
Docker storage	uses volumes and bind mounts to persist data even after a running container is stopped.
Docker Swarm	automates the deployment of containerized applications but was designed specifically to work with Docker Engine and other Docker tools making it a popular choice for teams already working in Docker environments.

Term	Definition
Docker	An open container platform for developing, shipping and running applications in containers.
Ecosystem	A composition of services, support and tools that are widely available. The Kubernetes ecosystem is a large, rapidly growing ecosystem where its services, support, and tools are widely available.
Enforceability (Control)	Istio provides control by enforcing policies across an entire fleet and ensures resources are fairly distributed among consumers.
Envoy proxy	All network traffic is subject to or intercepted by a proxy, called Envoy, used by the service mesh and allows many features depending on the configuration.
etcd	A highly available key value store that contains all the cluster data. For any deployment, the deployment configuration is stored in etcd. It is the source of truth for the state in a Kubernetes cluster, and the system works to bring the cluster state into line with what is stored in etcd.
Eviction	Process of terminating one or more Pods on Nodes.
Horizontal Pod Autoscaler	Also known as:HPA An API resource that automatically scales the number of Pod replicas based on targeted CPU utilization or custom metric targets.
Human operators	Understand the systems they control. They know how to deploy services and how to recognize and fix problems.
IBM Cloud catalog	provides various Services that range from visual recognition to natural language processing and creating chatbots.
IBM Cloud Container Registry	stores and distributes container images in a fully managed private registry.
Image Change	A trigger to rebuild a containerized application when a new or updated version of an image is available. For example, if an application is built using a Node.js base image, that image will be updated as security fixes are released and other updates occur.
Image	An immutable file that contains the source code, libraries, and dependencies that are necessary for an application to run. Images are templates or blueprints for a container.
ImageStream Tag	An identity to the pointer in an ImageStream that points to a certain image in a registry.
ImageStream	An abstraction for referencing container images within OpenShift. Each image contains an ID, or digest, that identifies it. ImageStreams do not contain image data but rather are pointers to image digests.

Term	Definition
Immutability	Images are read-only; if you change an image, you create a new image.
Imperative commands	Create, update, and delete live objects directly.
Imperative Management	Defining steps and actions to get to a desired state.
Ingress	An API object that manages external access to the services in a cluster, typically HTTP.
IPv4/IPv6 dual stack	Assigns both IPv4 and IPv6 addresses to Pods and Services.
Istio	A platform-independent and popular service mesh platform, often used with Kubernetes. It intelligently controls the flow of traffic and API calls between services, conducts a range of tests and reduces the complexity of managing network services. Istio secures services through authentication, authorization, and encryption. Istio provides control by defining policies that can be enforced across an entire fleet. With Istio, you can observe traffic flow in your mesh so you can trace call flows, dependencies, and you can view service communication metrics such as latency, traffic, errors and saturation.
Job	A finite or batch task that runs to completion.
kube-scheduler	Control plane component that watches for newly created Pods with no assigned node, and selects a node for them to run on.
Kubectl	Also known as kubectl Command line tool for communicating with a Kubernetes cluster's control plane, using the Kubernetes API.
Kubelet	The kubelet is the primary "node agent" that runs on each node. The kubelet takes a set of PodSpecs (a YAML or JSON object that describes a pod) provided primarily through the apiserver and ensures that the containers described in those PodSpecs are running and healthy. The kubelet doesn't manage containers which were not created by Kubernetes.
Kubernetes API Server	The Kubernetes API server validates and configures data for the api objects which include pods, services, replication controllers, and others. The API Server services REST operations and provides the frontend to the cluster's shared state through which all other components interact.
Kubernetes API	The application that serves Kubernetes functionality through a RESTful interface and stores the state of the cluster.
Kubernetes Cloud Controller Manager	A Kubernetes control plane component that embeds cloud-specific control logic. The cloud controller manager lets you link your cluster into your cloud provider's API, and separates out the components that interact with that cloud platform from components that only interact with your cluster.

Term	Definition
Kubernetes Controller Manager	Runs all the controller processes that monitor the cluster state and ensures that the actual state of a cluster matches the desired state. Examples of controllers that ship with Kubernetes are the replication controller, endpoints controller, namespace controller, and service accounts controller.
Kubernetes Proxy	A network proxy that runs on each node in a cluster. This proxy maintains network rules that allow communication to Pods running on nodes-in other words, communication to workloads running on the cluster. The user must create a service with the apiserver API to configure the proxy.
Kubernetes	is the de facto open-source platform standard for container orchestration. It was developed by Google and is maintained by the Cloud Native Computing Foundation (CNCF). Kubernetes automates container management tasks, like deployment, storage provisioning, load balancing and scaling, service discovery, and fixing failed containers. Its open-source toolset and wide array of functionalities are very attractive to leading cloud providers, who both support it, and in some cases, also offer fully managed Kubernetes services.
Label Selector	Allows users to filter a list of resources based on labels.
Labels	Tags objects with identifying attributes that are meaningful and relevant to users.
Linguistic Analysis	Detects the tone in a given text.
Load balancing	Balances traffic across Pods for better performance and high availability.
LXC	LinuX Containers is a OS-level virtualization technology that allows creation and running of multiple isolated Linux virtual environments (VE) on a single control host.
Man-in-the-middle attacks	A man-in-the-middle (MiTM) attack is a type of cyber-attack where the attacker secretly intercepts and relays messages between two parties who believe they are communicating directly with each other. The attack is a type of eavesdropping in which the attacker intercepts and then controls the entire conversation.
Marathon	is an Apache Mesos framework. Apache Mesos is an open-source cluster manager developed by UC Berkeley. It lets users scale container infrastructure through the automaton of most management and monitoring tasks.
Microservices	are a cloud-native architectural approach in which a single application contains many loosely coupled and independently deployable smaller components or services.

Term	Definition
Namespace	A Linux namespace is a Linux kernel feature that isolates and virtualizes system resources. Processes which are restricted to a namespace can only interact with resources or processes that are part of the same namespace. Namespaces are an important part of Docker's isolation model. Namespaces exist for each type of resource, including networking, storage, processes, hostname control and others.
Node	The worker machine in a Kubernetes cluster. User applications are run on nodes. Nodes can be virtual or physical machines. Each node is managed by the control plane and is able to run Pods.
Nomad	(Hashicorp) is a free and open-source cluster management and scheduling tool that supports Docker and other applications on all major operating systems across all infrastructure, whether on-premises or in the cloud. This flexibility lets teams work with any type and level of workload.
Object	An entity in the Kubernetes system. The Kubernetes API uses these entities to represent the state of your cluster.
Observability	Helps to observe the traffic flow in your mesh, trace call flows and dependencies, and view metrics such as latency and errors.
OpenShift CI/CD process	Automatically merges new code changes to the repository, builds, tests, approves, and deploys a new version to different environments.
OpenShift	A hybrid cloud, enterprise Kubernetes application.
Operating System Virtualization	OS-level virtualization is an operating system paradigm in which the kernel allows the existence of multiple isolated user space instances, called containers, zones, virtual private servers, partitions, virtual environments, virtual kernels, or jails.
Operator Framework	Is a family of tools and capabilities to deliver an efficient customer experience. It is not just about writing code; what is also critical is testing, delivery, and updating Operators.
Operator Lifecycle Manager	(or OLM) Controls the install, upgrade, and role-based access control (or RBAC) of Operators in a cluster.
Operator maturity model	Defines the phases of maturity for general day two Operations activities and ranges from Basic Install to Auto Pilot.
Operator Pattern	A system design that links a Controller to one or more custom resources.

Term	Definition
Operator Registry	Stores CRDs, cluster service versions (CSVs), and Operator metadata for packages and channels. It runs in Kubernetes or OpenShift clusters to provide the Operator catalog data to OLM.
Operator SDK	(which includes Helm, Go, and Ansible) Helps authors build, test, and package their Operators without requiring knowledge of Kubernetes API complexities.
OperatorHub	Web console lets cluster administrators find Operators to install on their cluster. It provides many different types of Operators available, including Red Hat Operators, Certified Operators from independent service vendors partnered with Red Hat, Community Operators from the open-source community but not officially supported by Red Hat, and custom Operators defined by users.
Operators	Automate cluster tasks and act as a custom controller to extend the Kubernetes API.
Persistence	Ensures that an object exists in the system, until the object is modified or removed.
Persistent Volume Claim	Claims storage resources defined in a PersistentVolume so that it can be mounted as a volume in a container.
Persistent Volume	An API object that represents a piece of storage in the cluster. Available as a general, pluggable resource that persists beyond the lifecycle of any individual Pod.
Pod	The smallest and simplest Kubernetes object. Represents a process running in a cluster; it also represents a single instance of an application running in a cluster. Usually, a Pod wraps a single container but, in some cases encapsulates multiple tightly coupled containers that share resources.
postCommit	Section defines an optional build hook.
Preemption	Logic in Kubernetes helps a pending Pod to find a suitable Node by evicting low priority Pods existing on that Node.
Private Registry	Restricts access to images so that only authorized users can view and use them.
Proxy	In computing, a proxy is a server that acts as an intermediary for a remote service.
Registry	is a hosted service containing repositories of images which responds to the Registry API.
ReplicaSet	A ReplicaSet (aims to) maintain a set of replica Pods running at any given time.

Term	Definition
Repository	is a set of Docker images. A repository can be shared by pushing it to a registry server. The different images in the repository can be labelled using tags.
REST API	A REST API (also known as RESTful API) is an application programming interface (API or web API) that conforms to the constraints of REST architectural style and allows for interaction with RESTful web services.
Retries	A method to prevent errors in one microservice from cascading to other microservices.
Rolling Updates	Provide a way to roll out application changes in an automated and controlled fashion throughout your pods. Rolling updates work with pod templates such as deployments. Rolling updates allow for rollback if something goes wrong.
runPolicy	Field controls how builds created from a build configuration need to run. Values include the default Serial (sequentially) and simultaneously.
Secrets	Stores sensitive information, such as passwords, OAuth tokens, and ssh keys.
Self-healing	Restarts, replaces, reschedules, and kills failing or unresponsive containers.
Server Virtualization	Server virtualization is the process of dividing a physical server into multiple unique and isolated virtual servers by means of a software application. Each virtual server can run its own operating systems independently.
Serverless	is a cloud-native development model that allows developers to build and run applications without having to manage servers.
Service binding	is the process needed to consume external Services or backing Services, including REST APIs, databases, and event buses in your applications.
Service Broker	Provides a short-running process that cannot perform the consecutive day's operations such as upgrades, failover, or scaling.
Service Discovery	Discovers Pods using their IP addresses or a single DNS name.
Service Mesh	A dedicated layer for making service-to-service communication secure and reliable. It provides traffic management to control the flow of traffic between services, security to encrypt traffic between services, and observability of service behavior; so, you can troubleshoot and optimize applications.
Service	An abstract way to expose an application running on a set of Pods as a network service.

Term	Definition
Software operators	Try to capture the knowledge of human operators and automate the same processes.
Source strategy	Section shows the strategy used to execute the build, such as a Source, Docker, or Custom strategy.
Source type	Determines the primary input like a Git repository, an inline Dockerfile, or binary payloads.
Source-to-Image	A tool for building reproducible container images. Also abbreviated S2i, it injects application source code into a container image to produce a ready-to-run image.
StatefulSet	Manages the deployment and scaling of a set of Pods, and provides guarantees about the ordering and uniqueness of these Pods.
Storage Orchestration	Automatically mounts your chosen storage system whether from local storage, network storage, or public cloud.
Storage	A data store that supports persistent and temporary storage for Pods.
Tag	A tag is a label applied to a Docker image in a repository. Tags are how various images in a repository are distinguished from each other.
Tone Analyzer Service	is used for explaining service binding. This IBM Cloud Service uses linguistic analysis to detect tone in a given text.
Vertical Pod Autoscaler also known as VPA	An API resource that adds resources to an existing machine. A VPA lets you scale a service vertically within a cluster.
Volume Mount	entails mounting of the declared volume into a container in the same Pod.
Volume Plugin	A Volume Plugin enables integration of storage within a Pod.
Volume	A directory containing data, accessible to multiple containers in a Pod.
Webhook	A trigger that sends a request to an OpenShift Container Platform API endpoint. Often this will be a GitHub webhook, though it can also be a generic webhook. If a GitHub webhook is utilized, GitHub can send the request to OpenShift when there is a new commit on a certain branch, or a pull request is merged, or under many more circumstances. Webhooks are a great way to automate development flows so that builds can occur automatically as new code is developed.
Workload	A workload is an application running on Kubernetes.

Cheat Sheet: Introduction to Containers w/ Docker, Kubernetes & OpenShift

Estimated reading time: 5 minutes

Command	Description
curl localhost	Pings the application.
docker build	Builds an image from a Dockerfile.
docker build . -t	Builds the image and tags the image id.
docker CLI	Start the Docker command line interface.
docker container rm	Removes a container.
docker images	Lists the images.
docker ps	Lists the containers.
docker ps -a	Lists the containers that ran and exited successfully.
docker pull	Pulls the latest image or repository from a registry.
docker push	Pushes an image or a repository to a registry.
docker run	Runs a command in a new container.
docker run -p	Runs the container by publishing the ports.
docker stop	Stops one or more running containers.
docker stop $(docker ps -q)	Stops all running containers.
docker tag	Creates a tag for a target image that refers to a source image.
docker -version	Displays the version of the Docker CLI.
exit	Closes the terminal session.
export MY_NAMESPACE	Exports a namespace as an environment variable.
for …do	Runs a for command multiple times as specified.
git clone	Clones the git repository that contains the artifacts needed.
ibmcloud cr images	Lists images in the IBM Cloud Container Registry.
ibmcloud cr login	Logs your local Docker daemon into IBM Cloud Container Registry.
ibmcloud cr namespaces	Views the namespaces you have access to.
ibmcloud cr region-set	Ensures that you are targeting the region appropriate to your cloud account.

Command	Description
ibmcloud target	Provides information about the account you're targeting.
ibmcloud version	Displays the version of the IBM Cloud CLI.
kubectl apply	Applies a configuration to a resource.
kubectl autoscale deployment	Autoscales a Kubernetes Deployment.
kubectl config get-clusters	Displays clusters defined in the kubeconfig.
kubectl config get-contexts	Displays the current context.
kubectl create	Creates a resource.
kubectl create configmap	Creates a ConfigMap resource.
kubectl delete	Deletes resources.
kubectl describe	Shows details of a resource or group of resources.
kubectl expose	Exposes a resource to the internet as a Kubernetes service.
kubectl get	Displays resources.
kubectl get deployments	Lists the deployments created.
kubectl get deployments -o wide	Lists deployments with details.
kubectl get hpa	Lists Horizontal Pod Autoscalers (hpa)
kubectl get pods	Lists all the Pods.
kubectl get pods -o wide	Lists all the Pods with details.
kubectl get services	Lists the services created.
kubectl proxy	Creates a proxy server between a localhost and the Kubernetes API server.
kubectl rollout	Manages the rollout of a resource.
kubectl rollout restart	Restarts the resource so that the containers restart.
kubectl rollout undo	Rollbacks the resource.
kubectl run	Creates and runs a particular image in a pod.
kubectl scale deployment	Scales a deployment.
kubectl set image deployment	Updates the current deployment.
kubectl version	Prints the client and server version information.
ls	Lists the contents of this directory to see the artifacts.
oc get	Displays a resource.

Command	Description
oc project	Switches to a different project.
oc version	Displays version information

Glossary: Application Development using Microservices and Serverless

Welcome! This alphabetized glossary contains many of the terms in this course. This comprehensive glossary also includes additional industry-recognized terms not used in course videos. These terms are essential for you to recognize when working in the industry, participating in user groups, and in other professional certificate programs.

Estimated reading time: 10 minutes

Term	Definition
API Gateway	API management tool that sits between a client and a collection of backend services
ASGI	Asynchronous Server Gateway Interface - web server interface to call microservice asynchronously
BaaS	Backend-as-a-Service
BFF	Backend for Frontend
Build	An executable unit
Buildpack	Contains executables to perform tasks such as inspecting source code, creating a build plan, or executing the build plan to produce a container image
CaaS	Containers as a Service
CNCF	Cloud Native Computing Foundation
Code Engine	Abstracts the operational burden of building, deploying, and managing workloads so that developers can focus on code development
Configuration	Everything that can differ between deployments, might differ among environments
Container	A standalone, all-inclusive, and executable unit of software packaged with libraries, dependencies, and runtimes
Container image	An immutable file containing all the application assets like source code, libraries, and dependencies necessary for an application to run
CRUD	Creating, reading, updating, and deleting records

Term	Definition
cURL	A command line tool and library for transferring data with URLs
DELETE	Request to delete a record
Docker	A software platform for building and running applications as containers
Dockerfile	A text file that includes all the commands to build a docker container image
Environment variables	Are easy to change across deployments without changing the code
FaaS	Function-as-a-Service
Flask	A micro web framework that does not require particular tools or libraries
GET	Request to retrieve a record
GraphQL	A query language that enables you to retrieve exactly what you need from the API
Horizontal scaling	Scaling by adding more instances of resources, also described as "scaling out"
IBM Cloud CLI	IBM Cloud command line interface
IBM Cloud Console	A well-designed web portal for end users to conveniently manage their IBM cloud services, including the Code Engine
IDE	Integrated Development Environment
Job	Runs executable code one time and exits
Microservices	A single application is composed of many loosely coupled and independently deployable smaller services
Monolithic application	Has all or most of its functionality within a single process
OpenAPI	specification Defines a standard, language-agnostic interface to RESTful APIs
PaaS	Platform as a Service
pip	Python package manager
POST	Request to create a record
Postman	An API platform for building and using APIs
PUT	Request to update a record

Term	Definition
Release stage	Combines the build with the deployment's current configuration so that the code is ready to run
Repository	A group of related container images
REST	Representational State Transfer
Run stage	Implements the application
SaaS	Software as a Service
Serverless computing	Abstracts both infrastructure and software environments. An architectural style where code runs using a cloud platform
Service discovery pattern	Helps applications and services discover each other
SOA	Service Oriented Architecture
SPA	Single page application
Stateless	Means each request contains all the information required to process it
Strangler pattern	Helps manage the refactoring of a monolithic application in stages
Swagger	Allows you to describe the structure of your APIs so that machines can read them
TCP	Transmission Control Protocol
TLS	Transport layer security
VM	Virtual machine
WebSocket	A communication protocol based on Transmission Control Protocol
WSGI	Web Server Gateway Interface - Python standard for communication between web servers and web applications or microservices