Code base Seminar

샘이나 샘이나~

Backend.ai 멘티 이유찬

Order

Step 1

Overview

Step 2

Client

Step 3

Manager

Step 4

Agent

01

OVERVIEW

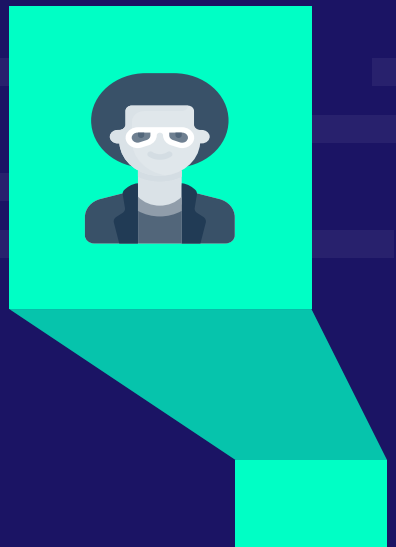
Backend.ai Tutorial

backend.ai run -c 'print("hello")' python

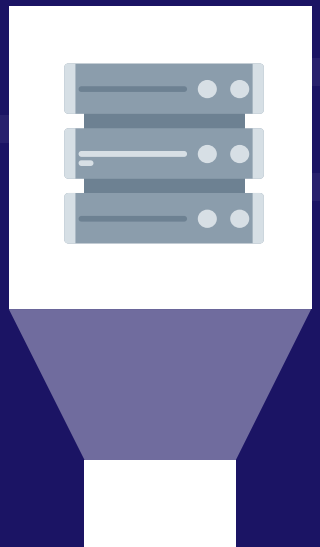
```
uchanlee — ubuntu@CA4: ~/uchan/backend.ai/backend.ai-dev — ssh ◂ sh enter.sh — 109×36

ream [2878292] STREAM_EXECUTE(ak:AKIAIOSFODNN7EXAMPLE,
 s:91100b1b-8301-468d-bd06-2f51c8a9599f)
2021-10-16 14:42:06.192 INFO aiohttp.access [2878292]
127.0.0.1 [16/Oct/2021:05:42:06 +0000] "GET /stream/se
ssion/91100b1b-8301-468d-bd06-2f51c8a9599f/execute HTT
P/1.1" 101 0 "-" "Backend.AI Client for Python 21.09.0
a2"

2021-10-16 14:42:06.181 INFO ai.backend.agent.server [
2886242] rpc::execute(k:91100b1b-8301-468d-bd06-2f51c8
a9599f, run-id:5b23b4c60834dbb0, mode:query, code:'pri
nt("hello")')
2021-10-16 14:42:06.190 DEBUG ai.backend.agent.agent [
2886242] _execute(91100b1b-8301-468d-bd06-2f51c8a9599f
) finished

1] started.
2021-10-16 14:41:10 INFO ai.backend.web.server [288975
3] started.
2021-10-16 14:41:10 INFO ai.backend.web.server [288975
2] started.
2021-10-16 14:41:10 INFO ai.backend.web.server [288975
4] started.

INFO Started service.
INFO Backend.AI Storage Proxy
INFO Runtime: Python 3.9.5 (env: /home/ubuntu/.pyenv/v
ersions/venv-4d9o0zwu-storage-proxy)
INFO Node ID: i-storage-proxy-01
DEBUG debug mode enabled.
INFO Using uvloop as the event loop backend

(venv-4d9o0zwu-client) ubuntu@CA4:~/uchan/backend.ai/backend.ai-dev/client-py$ backend.ai run -c 'print("hell
o")' python
· Session name prefix: pysdk-e197afd41d
✔ [0] Session pysdk-e197afd41d is ready (domain=default, group=default).
hello
✔ [0] Execution finished. (exit code = 0)
(venv-4d9o0zwu-client) ubuntu@CA4:~/uchan/backend.ai/backend.ai-dev/client-py$

[11] 0:user*                                                    "CA4" 14:41 16-Oct-21
```

Components

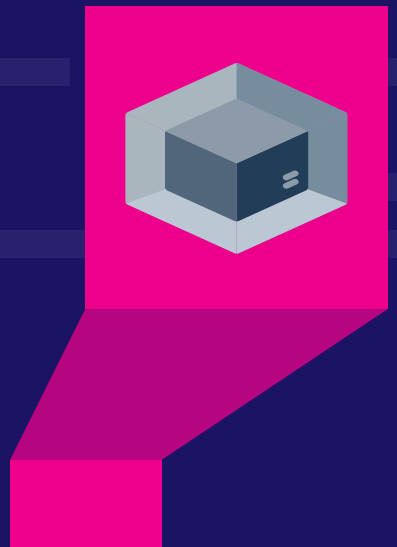
Client



Manager



Agent



02

Client

Client CLI

```
def _create_cmd(docs: str = None):

    @click.argument('image')
    @click.option('-t', '--name', '--client-token', metavar='NAME',
                  help='Specify a human-readable session name. '
                       'If not set, a random hex string is used.')
    @click.option('-o', '--owner', '--owner-access-key',
metavar='ACCESS_KEY',
                  help='Set the owner of the target session explicitly.')
    # job scheduling options
    @click.option('--type', metavar='SESSTYPE',
                  type=click.Choice(['batch', 'interactive']),
                  default='interactive',
                  help='Either batch or interactive')
    @click.option('--starts-at', metavar='STARTS_AT', type=str,
default=None,
                  help='Let session to be started at a specific or relative
time.')
    ...
```

```
def create(
        # base args
        image: str,
        name: str | None,
        owner: str | None,
        ...
        ...
    ) -> None:

        ...
        with Session() as session:
            try:
                compute_session = session.ComputeSession.get_or_create(
                    image,
                    name=name,
                    type_=type,
                    starts_at=starts_at,
                    enqueue_only=enqueue_only,
                    max_wait=max_wait,
                    no_reuse=no_reuse,
                    ...
```

Client Func(API)

```python
async def get_or_create(
    cls,
    image: str, *,
    name: str = None,
    type_: str = 'interactive',
    starts_at: str = None,
    enqueue_only: bool = False,
    max_wait: int = 0,
    no_reuse: bool = False,
    mounts: List[str] = None,
    mount_map: Mapping[str, str] = None,
    envs: Mapping[str, str] = None,
    startup_command: str = None,
    resources: Mapping[str, int] = None,
    resource_opts: Mapping[str, int] = None,
    cluster_size: int = 1,
    cluster_mode: Literal['single-node', 'multi-node'] = 'single-node',
    domain_name: str = None,
    ...,
) -> ComputeSession:
```

1. Set Param
2. Request & response

```python
rqst = Request('POST', f'/{prefix}')
    params: Dict[str, Any] = {
        'tag': tag,
        get_naming(api_session.get().api_version, 'name_arg'): name,
        'config': {
            ...
        },
    }
    if api_session.get().api_version >= (6, '20200815'):
        params['clusterSize'] = cluster_size
        params['clusterMode'] = cluster_mode
...

rqst.set_json(params)
    async with rqst.fetch() as resp:
        data = await resp.json()
        o = cls(name, owner_access_key)  # type: ignore
        if api_session.get().api_version[0] >= 5:
            o.id = UUID(data['sessionId'])
        ...
        return o
```

```
    async def get_or_create(
    cls,
    ...
    ...,
) -> ComputeSession:

    ...

    rqst = Request('POST', f'/{prefix}')
    params: Dict[str, Any] = {
        'tag': tag,
        get_naming(api_session.get().api_version, 'name_arg'): name,
        'config': {
            ...
        },
    }
    if api_session.get().api_version >= (6, '20200815'):
        params['clusterSize'] = cluster_size
        params['clusterMode'] = cluster_mode
    …

    rqst.set_json(params)
        async with rqst.fetch() as resp:
            data = await resp.json()
            o = cls(name, owner_access_key)  # type: ignore
            if api_session.get().api_version[0] >= 5:
                o.id = UUID(data['sessionId'])
        ...
            return o
```

Manager

```
    async def get_or_create(
    cls,
    ...
    ...,
) -> ComputeSession:

    ...

    rqst = Request('POST', f'/{prefix}')
    params: Dict[str, Any] = {
        'tag': tag,
        get_naming(api_session.get().api_version, 'name_arg'): name,
        'config': {
            ...
        },
    }
    if api_session.get().api_version >= (6, '20200815'):
        params['clusterSize'] = cluster_size
        params['clusterMode'] = cluster_mode
    ...

    rqst.set_json(params)
            async with rqst.fetch() as resp:
                data = await resp.json()
                o = cls(name, owner_access_key)  # type: ignore
                if api_session.get().api_version[0] >= 5:
                    o.id = UUID(data['sessionId'])
            ...
                return o
```

03

Manager

Manager API session

```
async def _create(request: web.Request, params: Any) ->
web.Response:
    if params['domain'] is None:
        params['domain'] = request['user']['domain_name']
    scopes_param = {
        'owner_access_key': (None if params['owner_access_key'] is
undefined
                             else params['owner_access_key'])
    }
    requester_access_key, owner_access_key = await
get_access_key_scopes(request, scopes_param)
    log.info('GET_OR_CREATE (ak:{0}/{1}, img:{2}, s:{3})',
             requester_access_key, owner_access_key if
owner_access_key != requester_access_key else '*',
             params['image'], params['session_name'])

    root_ctx: RootContext = request.app['_root.context']
    ...
```

```
...
# Resolve the image reference.
    try:
        requested_image_ref = \
            await ImageRef.resolve_alias(params['image'],
root_ctx.shared_config.etcd)
        async with root_ctx.db.begin() as conn:
            query = (sa.select([domains.c.allowed_docker_registries])
                     .select_from(domains)
                     .where(domains.c.name == params['domain']))
            allowed_registries = await conn.scalar(query)
            if requested_image_ref.registry not in allowed_registries:
                raise AliasResolutionFailed
    except AliasResolutionFailed:
        raise ImageNotFound('unknown alias or disallowed registry')

...
```

Manager API session

```
...
# Resolve the image reference.
    try:
        requested_image_ref = \
            await ImageRef.resolve_alias(params['image'],
root_ctx.shared_config.etcd)
        async with root_ctx.db.begin() as conn:
            query = (sa.select([domains.c.allowed_docker_registries])
                     .select_from(domains)
                     .where(domains.c.name == params['domain']))
            allowed_registries = await conn.scalar(query)
            if requested_image_ref.registry not in allowed_registries:
                raise AliasResolutionFailed
    except AliasResolutionFailed:
        raise ImageNotFound('unknown alias or disallowed registry')

...
```

```
try:

        kernel_id = await
asyncio.shield(root_ctx.registry.enqueue_session(
            session_creation_id,
            params['session_name'], owner_access_key,
            [{
                'image_ref': requested_image_ref,
                ...
            }],
            params['config']['scaling_group'],
            params['session_type'],


            ...
        ))
        resp['sessionId'] = str(kernel_id)  # changed since API v5
        resp['sessionName'] = str(params['session_name'])
        resp['status'] = 'PENDING'
        resp['servicePorts'] = []
        resp['created'] = True
```

Manager registry

```
async def enqueue_session(
        self,
        session_creation_id: str,
        session_name: str,
        ...
    ) -> SessionId:

        mounts =
kernel_enqueue_configs[0]['creation_config'].get('mounts') or []
        mount_map =
kernel_enqueue_configs[0]['creation_config'].get('mount_map') or {}
        session_id = SessionId(uuid.uuid4())
        ...
```

```
...
is_multicontainer = cluster_size > 1
if is_multicontainer:
    if len(kernel_enqueue_configs) == 1:
        log.debug(
            'enqueue_session(): replicating kernel_enqueue_config with
cluster_size={}',
            cluster_size,
        )
        # the first kernel_config is repliacted to sub-containers
        assert kernel_enqueue_configs[0]['cluster_role'] ==
DEFAULT_ROLE
        kernel_enqueue_configs[0]['cluster_idx'] = 1
        for i in range(cluster_size - 1):
            sub_kernel_config = cast(KernelEnqueueingConfig,
{**kernel_enqueue_configs[0]})
            sub_kernel_config['cluster_role'] = 'sub'
            sub_kernel_config['cluster_idx'] = i + 1
            kernel_enqueue_configs.append(sub_kernel_config)
    ...
...
```

Manager registry

```
for kernel in kernel_enqueue_configs:
        kernel_id: KernelId
        if kernel['cluster_role'] == DEFAULT_ROLE:
            kernel_id = cast(KernelId, session_id)
        else:
            kernel_id = KernelId(uuid.uuid4())
        creation_config = kernel['creation_config']
        image_ref = kernel['image_ref']
        ...
        image_info = await self.shared_config.inspect_image(image_ref)
        image_min_slots, image_max_slots = \
            await self.shared_config.get_image_slot_ranges(image_ref)
                # Parse service ports to check for port errors
        parse_service_ports(image_info['labels'].get('ai.backend.service-ports', ''),
BackendError)
        ...
        # Check if: requested <= image-maximum
        if not (requested_slots <= image_max_slots):
            raise InvalidAPIParameters(...)
        environ = kernel_enqueue_configs[0]['creation_config'].get('environ') or {}
```

```
async def _enqueue() -> None:
            nonlocal ids
            async with self.db.begin() as conn:
                query = kernels.insert().values({
                    'id': kernel_id,
                    'status': KernelStatus.PENDING,
                    'session_creation_id': session_creation_id,
                    'session_id': session_id,
                    'session_name': session_name,
                    ...
                    'user_uuid': user_uuid,
                    'access_key': access_key,
                    'image': image_ref.canonical,
                    'registry': image_ref.registry,
                    'tag': session_tag,
                    ...
                })
                await conn.execute(query)
                ids.append(kernel_id)

        await execute_with_retry(_enqueue)
```

Manager API session

```
async def _create_kernels_in_one_agent(
    self,
    agent_alloc_ctx: AgentAllocationContext,
    scheduled_session: PendingSession,
    items: Sequence[KernelAgentBinding],
    image_info,
    cluster_info,
  ) -> None:
    ...
    async with RPCContext(...) as rpc:
      kernel_creation_id = secrets.token_urlsafe(16)
      # Prepare kernel_started event handling
      for binding in items:
        self.kernel_creation_tracker[
          binding.kernel.kernel_id
        ] = loop.create_future()
```

```
try:
        # Issue a batched RPC call to create kernels on this agent
        created_infos = await rpc.call.create_kernels(...)

        # Post-process kernel creation
        async with aiotools.TaskGroup() as tg:
          for created_info in created_infos:
            post_task = tg.create_task(self._post_create_kernel(
              agent_alloc_ctx,
              created_info,
            ))
            self._post_kernel_creation_tasks[created_info['id']] =
post_task
      except Exception:
        ...
      finally:
        # clean up for sure
        for binding in items:
          del self.kernel_creation_tracker[
            binding.kernel.kernel_id
          ]
```

Manager API session

```python
async def _post_create_kernel(
    self,
    agent_alloc_ctx: AgentAllocationContext,
    created_info,
):
    # Wait until the kernel_started event.
    kernel_id = KernelId(uuid.UUID(created_info['id']))
    start_event = self.kernel_creation_tracker[kernel_id]
    try:
        await start_event
    except asyncio.CancelledError:
        log.warning("post_create_kernel(k:{}) cancelled", kernel_id)
        return
    # Record kernel access information
```

```python
async def _finalize_running() -> None:
    async with self.db.begin() as conn:
        agent_host = URL(agent_alloc_ctx.agent_addr).host
        kernel_host = created_info.get('kernel_host', agent_host)
        service_ports = created_info.get('service_ports', [])
        # NOTE: created_info contains resource_spec
        query = (
            kernels.update()
            .values({
                'scaling_group': agent_alloc_ctx.scaling_group,
                'status': KernelStatus.RUNNING,
                'container_id': created_info['container_id'],
                ...
            })
            .where(kernels.c.id == created_info['id']))
        await conn.execute(query)

    await execute_with_retry(_finalize_running)
```

session_name	status	kernel_id	···
psdk-1a..	PENDING	a8411..	···

Manager

Agent

04

Agent

Agent agent

```python
async def create_kernel(
        self,
        creation_id: str,
        session_id: SessionId,
        kernel_id: KernelId,
        kernel_config: KernelCreationConfig,
        cluster_info: ClusterInfo,
        *,
        restarting: bool = False,
    ) -> KernelCreationResult:
        """
        Create a new kernel.
        """

        ...
        # Finally we are done.
        await self.produce_event(
            KernelStartedEvent(kernel_id, creation_id)
        )
```

```python
 if kernel_config['session_type'] == 'batch' and
kernel_config['cluster_role'] == 'main':
            self._ongoing_exec_batch_tasks.add(
                asyncio.create_task(
                    self.execute_batch(kernel_id,
kernel_config['startup_command'] or "")
                )
            )
        return {
            'id': KernelId(kernel_id),
            'kernel_host': str(kernel_obj['kernel_host']),
            'repl_in_port': kernel_obj['repl_in_port'],
            'repl_out_port': kernel_obj['repl_out_port'],
            'stdin_port': kernel_obj['stdin_port'],     # legacy
            'stdout_port': kernel_obj['stdout_port'],   # legacy
            'service_ports': service_ports,
            'container_id': kernel_obj['container_id'],
            'resource_spec': resource_spec.to_json_serializable_dict(),
            'attached_devices': attached_devices,
        }
```

Agent docker agent

```
    async with closing_async(Docker()) as docker:
        try:
            container = await docker.containers.create(
                config=container_config, name=kernel_name)
            cid = container._id

            resource_spec.container_id = cid
            # Write resource.txt again to update the contaienr id.
            with open(self.config_dir / 'resource.txt', 'w') as f:
                await loop.run_in_executor(None, resource_spec.write_to_file, f)
            async with AsyncFileWriter(
                target_filename=self.config_dir / 'resource.txt',
                access_mode='a',
            ) as writer:
                for dev_name, device_alloc in resource_spec.allocations.items():
                    computer_ctx = self.computers[dev_name]
                    kvpairs = \
                        await
computer_ctx.instance.generate_resource_data(device_alloc)
                    for k, v in kvpairs.items():
                        await writer.write(f'{k}={v}\n')

            await container.start()
```

```
kernel_obj = await DockerKernel.new(
        self.kernel_id,
        self.image_ref,
        self.kspec_version,
        agent_config=self.local_config,
        service_ports=service_ports,
        resource_spec=resource_spec,
        data={
            'container_id': container._id,
            'kernel_host': advertised_kernel_host or container_bind_host,
            'repl_in_port': repl_in_port,
            'repl_out_port': repl_out_port,
            'stdin_port': stdin_port,    # legacy
            'stdout_port': stdout_port,  # legacy
            'host_ports': host_ports,
            'domain_socket_proxies': self.domain_socket_proxies,
            'block_service_ports': self.internal_data.get('block_service_ports', False),
        })
    return kernel_obj
```

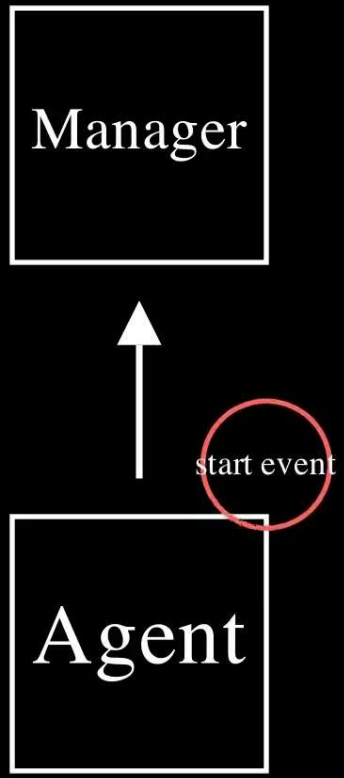
Agent docker agent

```python
class DockerAgent(AbstractAgent[DockerKernel,
DockerKernelCreationContext]):

    monitor_docker_task: asyncio.Task
    agent_sockpath: Path
    agent_sock_task: asyncio.Task
    scan_images_timer: asyncio.Task

    def __init__(
        self,
        etcd: AsyncEtcd,
        local_config: Mapping[str, Any],
        *,
        stats_monitor: StatsPluginContext,
        error_monitor: ErrorPluginContext,
        skip_initial_scan: bool = False,
    ) -> None:
        ...
```

Agent Agent

```python
@abstractmethod
    async def pull_image(self, image_ref: ImageRef, registry_conf:
ImageRegistry) -> None:
        '''
        Pull the given image from the given registry.
        '''
```

```python
async def pull_image(self, image_ref: ImageRef, registry_conf:
ImageRegistry) -> None:
    auth_config = None
    reg_user = registry_conf.get('username')
    reg_passwd = registry_conf.get('password')
    if reg_user and reg_passwd:
        encoded_creds = base64.b64encode(
            f'{reg_user}:{reg_passwd}'.encode('utf-8')) \
            .decode('ascii')
        auth_config = {
            'auth': encoded_creds,
        }
    log.info('pulling image {} from registry', image_ref.canonical)
    async with closing_async(Docker()) as docker:
        await docker.images.pull(
            image_ref.canonical,
            auth=auth_config)
```

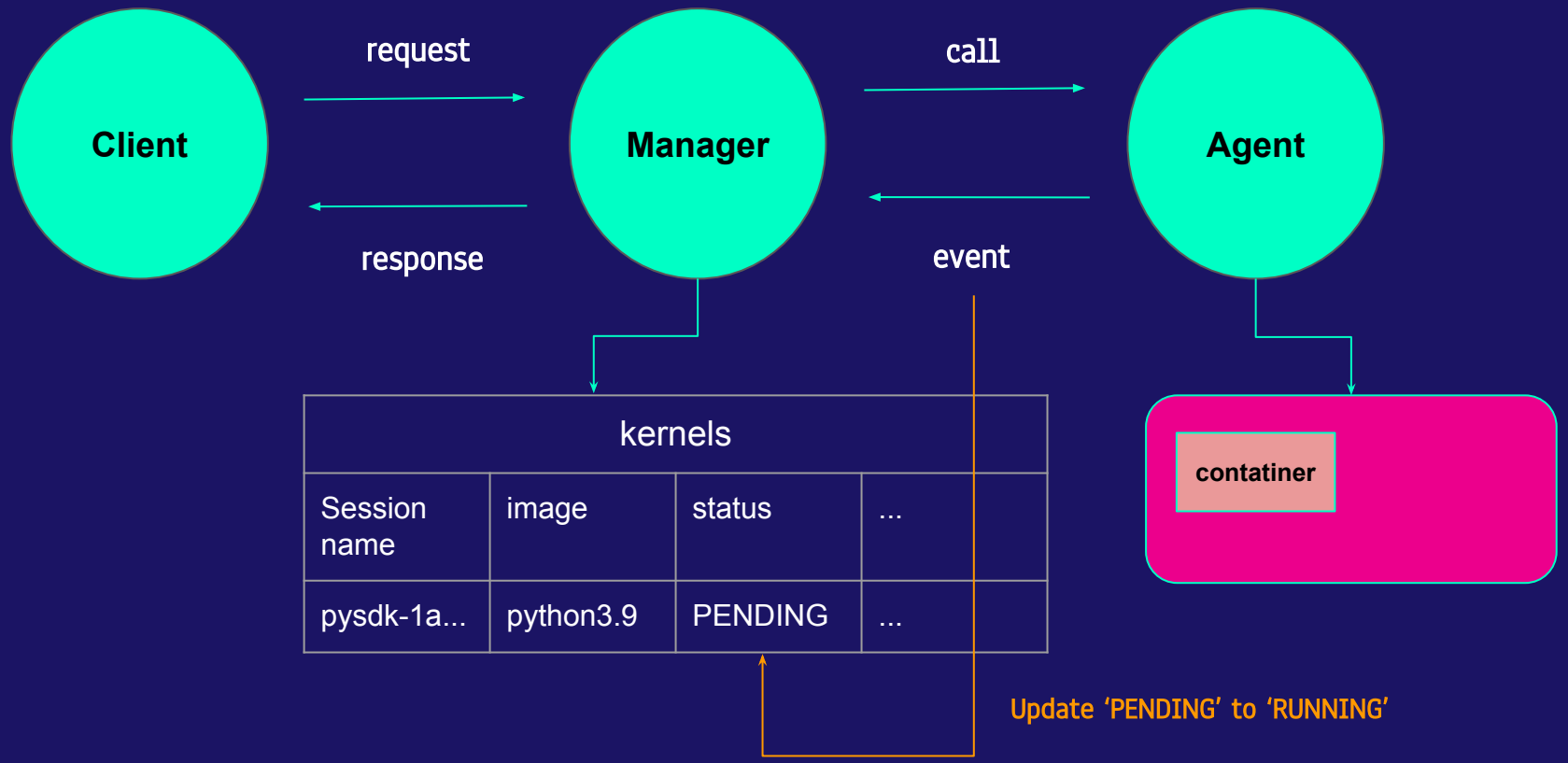

Manager
start event
Agent


AGENT DOCKER
container

AGENT DOCKER

Agent

Summary


Client
request
response
Manager
call
event
Agent
kernels
Session name
image
status
...
pysdk-1a...
python3.9
PENDING
...
contatiner
Update 'PENDING' to 'RUNNING'

THANKS !