Aliyun Elastic IPs verwalten

Dieses Skript bietet eine Kommandozeilen-Schnittstelle zur Verwaltung von Aliyun Elastic IPs (EIPs). Es ermöglicht das Erstellen, Binden, Aufheben der Bindung und Freigeben von EIPs mithilfe des Aliyun SDK für Python. Das Skript verwendet Argumente für den auszuführenden Job und die Allocation ID der EIP.

```
python aliyun_elastic_ip_manager.py unbind --allocation_id eip-j6c2olvsa7jk9l42iaaa
python aliyun_elastic_ip_manager.py bind --allocation_id eip-j6c7mhenamvy6zao3haaa
python aliyun_elastic_ip_manager.py release --allocation_id eip-j6c2olvsa7jk9l42aaa
python aliyun_elastic_ip_manager.py describe
```

```
# -*- coding: utf-8 -*-
# Diese Datei wurde automatisch generiert, bearbeiten Sie sie nicht. Danke.
import logging
import os
import sys
from typing import List
import argparse
import json

from alibabacloud_vpc20160428.client import Client as Vpc20160428Client
from alibabacloud_tea_openapi import models as open_api_models
from alibabacloud_vpc20160428 import models as vpc_20160428_models
from alibabacloud_tea_util import models as util_models
from alibabacloud_tea_util.client import Client as UtilClient
from alibabacloud_ecs20140526.client import Client as Ecs20140526Client



logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s - %(message)s')

class Sample:
    def __init__(self):
        pass

    @staticmethod
    def create_client() -> Vpc20160428Client:
        config = open_api_models.Config(
            access_key_id=os.environ['ALIBABA_CLOUD_ACCESS_ID_API_KEY'],
            access_key_secret=os.environ['ALIBABA_CLOUD_ACCESS_API_KEY']
        )
        config.endpoint = f'vpc.cn-hongkong.aliyuncs.com'
```

```
    return Vpc20160428Client(config)

@staticmethod
def bind_eip(
    region_id: str,
    allocation_id: str,
    instance_id: str,
) -> bool:
    client = Sample.create_client()
    associate_eip_address_request = vpc_20160428_models.AssociateEipAddressRequest(
        region_id=region_id,
        allocation_id=allocation_id,
        instance_id=instance_id
    )
    runtime = util_models.RuntimeOptions(read_timeout=60000, connect_timeout=60000)
    try:
        result = client.associate_eip_address_with_options(associate_eip_address_request, runtime)
        logging.info(f"EIP {allocation_id} erfolgreich an Instanz {instance_id} gebunden. Ergebnis: {resul
        return True
    except Exception as error:
        logging.error(f"Fehler beim Binden von EIP {allocation_id} an Instanz {instance_id}: {error}")
        if hasattr(error, 'message'):
            logging.error(f"Fehlermeldung: {error.message}")
        if hasattr(error, 'data') and error.data and error.data.get('Recommend'):
            logging.error(f"Empfehlung: {error.data.get('Recommend')}")
        UtilClient.assert_as_string(str(error))
        return False

@staticmethod
def unbind_eip(
    region_id: str,
    allocation_id: str,
    instance_id: str,
) -> bool:
    client = Sample.create_client()
    unassociate_eip_address_request = vpc_20160428_models.UnassociateEipAddressRequest(
        region_id=region_id,
        allocation_id=allocation_id,
        instance_id=instance_id
    )
```

```
    runtime = util_models.RuntimeOptions(read_timeout=60000, connect_timeout=60000)
    try:
        result = client.unassociate_eip_address_with_options(unassociate_eip_address_request, runtime)
        logging.info(f"EIP {allocation_id} erfolgreich von Instanz {instance_id} getrennt. Ergebnis: {resu
        return True
    except Exception as error:
        logging.error(f"Fehler beim Trennen von EIP {allocation_id} von Instanz {instance_id}: {error}")
        if hasattr(error, 'message'):
            logging.error(f"Fehlermeldung: {error.message}")
        if hasattr(error, 'data') and error.data and error.data.get('Recommend'):
            logging.error(f"Empfehlung: {error.data.get('Recommend')}")
        UtilClient.assert_as_string(str(error))
        return False

@staticmethod
def create_eip(
    region_id: str,
) -> str | None:
    client = Sample.create_client()
    allocate_eip_address_request = vpc_20160428_models.AllocateEipAddressRequest(
        region_id= region_id,
        instance_charge_type='PostPaid',
        internet_charge_type='PayByTraffic',
        bandwidth='200'
    )
    runtime = util_models.RuntimeOptions(read_timeout=60000, connect_timeout=60000)
    try:
        result = client.allocate_eip_address_with_options(allocate_eip_address_request, runtime)
        print(result.body)
        allocation_id = result.body.allocation_id
        logging.info(f"EIP erfolgreich erstellt. Allocation ID: {allocation_id}")
        return allocation_id
    except Exception as error:
        logging.error(f"Fehler beim Erstellen der EIP: {error}")
        if hasattr(error, 'message'):
            logging.error(f"Fehlermeldung: {error.message}")
        if hasattr(error, 'data') and error.data and error.data.get('Recommend'):
            logging.error(f"Empfehlung: {error.data.get('Recommend')}")
        UtilClient.assert_as_string(str(error))
        return None
```

```python
@staticmethod
def release_eip(
    allocation_id: str,
) -> bool:
    client = Sample.create_client()
    release_eip_address_request = vpc_20160428_models.ReleaseEipAddressRequest(
        allocation_id=allocation_id
    )
    runtime = util_models.RuntimeOptions(read_timeout=60000, connect_timeout=60000)
    try:
        result = client.release_eip_address_with_options(release_eip_address_request, runtime)
        logging.info(f"EIP {allocation_id} erfolgreich freigegeben. Ergebnis: {result}")
        return True
    except Exception as error:
        logging.error(f"Fehler beim Freigeben von EIP {allocation_id}: {error}")
        if hasattr(error, 'message'):
            logging.error(f"Fehlermeldung: {error.message}")
        if hasattr(error, 'data') and error.data and error.data.get('Recommend'):
            logging.error(f"Empfehlung: {error.data.get('Recommend')}")
        UtilClient.assert_as_string(str(error))
        return False


@staticmethod
def describe_eip(
    region_id: str,
    instance_id: str,
) -> str | None:
    client = Sample.create_client()
    describe_eip_addresses_request = vpc_20160428_models.DescribeEipAddressesRequest(
        region_id=region_id
    )
    runtime = util_models.RuntimeOptions(read_timeout=60000, connect_timeout=60000)
    try:
        result = client.describe_eip_addresses_with_options(describe_eip_addresses_request, runtime)
        logging.info(f"EIP erfolgreich beschrieben.")
        print(json.dumps(result.body.to_map(), indent=4))
        if result.body.eip_addresses and hasattr(result.body.eip_addresses, 'eip_address') and result.body
            for eip in result.body.eip_addresses.eip_address:
                if eip.instance_id == instance_id:
```

```
                    return eip.allocation_id
            logging.info(f"Keine EIP-Adresse für Instanz {instance_id} gefunden")
            return None
        else:
            logging.info("Keine EIP-Adressen gefunden.")
            return None
    except Exception as error:
        logging.error(f"Fehler beim Beschreiben der EIP: {error}")
        if hasattr(error, 'message'):
            logging.error(f"Fehlermeldung: {error.message}")
        if hasattr(error, 'data') and error.data and error.data.get('Recommend'):
            logging.error(f"Empfehlung: {error.data.get('Recommend')}")
        UtilClient.assert_as_string(str(error))
        return None

@staticmethod
def main(
    args: List[str],
) -> None:
    region_id = "cn-hongkong"
    instance_id = "i-j6c44l4zpphv7u7agdbk"

    parser = argparse.ArgumentParser(description='Aliyun Elastic IPs verwalten.')
    parser.add_argument('job', choices=['create', 'bind', 'unbind', 'release', 'describe', 'all'], help='I
    parser.add_argument('--allocation_id', type=str, help='Die Allocation ID der EIP.')
    parser.add_argument('--instance_id', type=str, default=instance_id, help='Die Instanz-ID, an die die I

    parsed_args = parser.parse_args(args)

    if parsed_args.job == 'create':
        new_allocation_id = Sample.create_eip(region_id)
        if new_allocation_id:
            print(f"EIP-Erstellungsprozess erfolgreich initiiert. Allocation ID: {new_allocation_id}")
        else:
            print("EIP-Erstellungsprozess fehlgeschlagen.")
    elif parsed_args.job == 'bind':
        if not parsed_args.allocation_id:
            print("Fehler: --allocation_id wird für den bind-Job benötigt.")
            return
        if Sample.bind_eip(region_id, parsed_args.allocation_id, parsed_args.instance_id):
```

```
            print(f"EIP-Bindungsprozess erfolgreich für EIP {parsed_args.allocation_id} und Instanz {parse
        else:
            print(f"EIP-Bindungsprozess für EIP {parsed_args.allocation_id} und Instanz {parsed_args.insta
    elif parsed_args.job == 'unbind':
        if not parsed_args.allocation_id:
            print("Fehler: --allocation_id wird für den unbind-Job benötigt.")
            return
        if Sample.unbind_eip(region_id, parsed_args.allocation_id, parsed_args.instance_id):
            print(f"EIP-Trennungsprozess erfolgreich für EIP {parsed_args.allocation_id} und Instanz {pars
        else:
            print(f"EIP-Trennungsprozess für EIP {parsed_args.allocation_id} und Instanz {parsed_args.inst
    elif parsed_args.job == 'release':
        if not parsed_args.allocation_id:
            print("Fehler: --allocation_id wird für den release-Job benötigt.")
            return
        if Sample.release_eip(parsed_args.allocation_id):
             print(f"EIP-Freigabeprozess erfolgreich für EIP {parsed_args.allocation_id} initiiert.")
        else:
            print(f"EIP-Freigabeprozess für EIP {parsed_args.allocation_id} fehlgeschlagen.")
    elif parsed_args.job == 'describe':
        if not parsed_args.instance_id:
            print("Fehler: --instance_id wird für den describe-Job benötigt.")
            return
        allocation_id = Sample.describe_eip(region_id, parsed_args.instance_id)
        if allocation_id:
            print(f"Allocation ID: {allocation_id}")
        else:
            print("Keine EIP gefunden.")
    elif parsed_args.job == 'all':
        # Beschreiben, um die aktuelle Allocation ID zu erhalten
        current_allocation_id = Sample.describe_eip(region_id, parsed_args.instance_id)
        if current_allocation_id:
            print(f"Aktuelle Allocation ID: {current_allocation_id}")
        else:
            print("Keine EIP zum Verarbeiten gefunden.")
            return

        # Aktuelle EIP trennen
        if current_allocation_id:
            if Sample.unbind_eip(region_id, current_allocation_id, parsed_args.instance_id):
```

```
                print(f"EIP {current_allocation_id} erfolgreich von Instanz {parsed_args.instance_id} getr
            else:
                print(f"EIP {current_allocation_id} konnte nicht von Instanz {parsed_args.instance_id} get
                return

        # Neue EIP erstellen
        new_allocation_id = Sample.create_eip(region_id)
        if new_allocation_id:
            print(f"EIP-Erstellungsprozess erfolgreich initiiert. Neue Allocation ID: {new_allocation_id}"
        else:
            print("EIP-Erstellungsprozess fehlgeschlagen.")
            return

        # Neue EIP binden
        if Sample.bind_eip(region_id, new_allocation_id, parsed_args.instance_id):
            print(f"Neue EIP {new_allocation_id} erfolgreich an Instanz {parsed_args.instance_id} gebunden
        else:
            print(f"Neue EIP {new_allocation_id} konnte nicht an Instanz {parsed_args.instance_id} gebunde
            return

        # Alte EIP freigeben
        if current_allocation_id:
            if Sample.release_eip(current_allocation_id):
                print(f"Alte EIP {current_allocation_id} erfolgreich freigegeben.")
            else:
                print(f"Alte EIP {current_allocation_id} konnte nicht freigegeben werden.")

        # Erneut beschreiben, um den endgültigen Zustand anzuzeigen
        final_allocation_id = Sample.describe_eip(region_id, parsed_args.instance_id)
        if final_allocation_id:
            print(f"Endgültige Allocation ID: {final_allocation_id}")
        else:
            print("Nach der Verarbeitung wurde keine EIP gefunden.")


@staticmethod
async def main_async(
    args: List[str],
) -> None:
    client = Sample.create_client()
```

```
        associate_eip_address_request = vpc_20160428_models.AssociateEipAddressRequest(
            region_id='cn-hongkong'
        )
        runtime = util_models.RuntimeOptions()
        try:
            await client.associate_eip_address_with_options_async(associate_eip_address_request, runtime)
        except Exception as error:
            print(error)
            if hasattr(error, 'message'):
                print(error.message)
            if hasattr(error, 'data') and error.data.get("Recommend"):
                print(error.data.get("Recommend"))
            UtilClient.assert_as_string(str(error))


if __name__ == '__main__':
    Sample.main(sys.argv[1:])

# python scripts/auto-ss-config/aliyun_elastic_ip_manager.py create
# python scripts/auto-ss-config/aliyun_elastic_ip_manager.py unbind --allocation_id eip-j6c2olvsa7jk9l42i1aaa
# python scripts/auto-ss-config/aliyun_elastic_ip_manager.py bind --allocation_id eip-j6c7mhenamvy6zao3haaa
# python scripts/auto-ss-config/aliyun_elastic_ip_manager.py release --allocation_id "eip-j6c2olvsa7jk9l42i"
# python scripts/auto-ss-config/aliyun_elastic_ip_manager.py describe
# python scripts/auto-ss-config/aliyun_elastic_ip_manager.py all
```